Visual Studio ALM Rangers

Visual Studio 2010 データベース プロジェクト ガイド

2010/10/12

Visual Studio ALM Rangers

Microsoft Corporation

Visual Studio ALM Rangers

このコンテンツは、Visual Studio 製品チーム、Microsoft Services、Microsoft MVP、および Visual Studio コミュニティ リーダーで構成された Visual Studio ALM Rangers によって作成されました。

本ドキュメントは『Visual Studio 2010 Database Projects Guidance Document』の日本語訳です。
原文は以下の URL から入手いただけます。

http://vsdatabaseguide.codeplex.com/releases/view/50499

目次

画像の目次

寄稿者

Shishir Abhyanker	マイクロソフト、シニア開発エンジニア II | Visual Studio ALM Ranger
Chris Burrows	マイクロソフト、シニア コンサルタント
David V Corbin	MVP
Larry Guger	MVP、Visual Studio ALM Ranger
Barclay Hill	マイクロソフト、シニア プログラム マネージャー、Visual Studio Business Applications チーム
Bob Leithiser	マイクロソフト、プリンシパル コンサルタント
Pablo Rincon	マイクロソフト、ソフトウェア開発エンジニア | Visual Studio ALM Ranger
Scott Sharpe	マイクロソフト、シニア ソフトウェア開発エンジニア | Visual Studio ALM Ranger
Jens K. Süßmeyer	マイクロソフト、シニア コンサルタント | Visual Studio ALM Ranger
LeRoy Tuttle	マイクロソフト、シニア開発エンジニア | Visual Studio ALM Ranger
Ryan Frazier	マイクロソフト、シニア コンサルタント

レビューアー

Visual Studio ALM Rangers	Visual Studio ALM Core and Extended Rangers
Christian Bitter	マイクロソフト、コンサルタント
Regis Gimenis	マイクロソフト、プリンシパル コンサルタント
Rob Jarrat	マイクロソフト、プリンシパル コンサルタント II
Bijan Javidi	マイクロソフト、プリンシパル アーキテクト | Visual Studio ALM Ranger リーダー
Willy-Peter Schaub	マイクロソフト、ソリューション アーキテクト | Visual Studio ALM Ranger
Mathias Olausson	Visual Studio ALM Ranger

このガイダンスは、製品に付属しているドキュメントおよび http://msdn.microsoft.com からアクセスできる Microsoft Developer Network (MSDN) と併用してください。

はじめに

概要

「Visual Studio 2010 データベース プロジェクト ガイド」をご覧いただき、ありがとうございます。

このガイドをお読みいただくと、Visual Studio 2010 でデータベース プロジェクトを使用するシナリオと手法を理解し、Visual Studio をより効果的に利用して、その価値を最大限に高められるようになります。

MSDN Web サイトで公開されているドキュメントでは、Visual Studio データベース プロジェクトの基礎知識と使用法が解説されていますが、実経験から得られる、テクノロジの理想的な使用法についてのガイダンスと推奨事項が含まれていないことがあります。製品やプラットフォームの機能や制限により、目的を達成するために特殊な処理が必要になることがあります。このガイドは、MSDN で公開されている情報を補完して、Visual Studio データベース プロジェクトをより効率的に使用できるようにサポートすることを目的としています。

このガイドには、既存のソリューションのリバース エンジニアリングや、ゼロから新しいソリューションを作成するなどの一般的なシナリオを提供しています。このようなシナリオは、アナリスト、アーキテクト、または開発者が実世界で直面する課題です。なお、このガイダンスは、製品の機能を詳細に解説することではなく、一般的なシナリオ、実用的なガイダンス、およびチェックリストを提供することを目的としています。

Visual Studio ALM Rangers

このガイドのコンテンツは、Visual Studio ALM Rangers のプロジェクトで作成されました。Visual Studio ALM Rangers は、Visual Studio 製品グループ、Microsoft Services、マイクロソフト MVP、および Visual Studio コミュニティのリーダーで構成されたグループです。チームの使命は、不足している機能の既定外ソリューションとガイダンスを提供することです。

このガイドは、データベース プロジェクトに関して、マイクロソフトが定める "200 ～ 300 レベル" のユーザーを対象としています (実環境における製品の機能を詳細に理解している、データベース プロジェクトの中級ユーザーから上級ユーザーを対象としています)。主な対象読者ではありませんが、このガイドには、データベース プロジェクトの初心者や専門家の方々に役立つ情報も含まれています。

一般的なガイダンス

目的

- Visual Studio データベース プロジェクトの理念の概要を説明する
- このガイドで紹介するトピックの概要を説明する
- 目的に応じて、読み始める場所を提案する

概要

Visual Studio データベース プロジェクトは、Visual Studio の標準プロジェクト テンプレートとして組み込まれているので、データベース開発作業の管理を、組織が他の Visual Studio プロジェクトの種類のために確立したプロセスとワークフローに統合できます。その結果、組織では、より総合的な ALM プロセスを実現できます。

注意

新しい Visual Studio データベース プロジェクト (プロジェクトのダイアログ ボックスで [データベース プロジェクト] をクリックし、SQL Server の適切なバージョンをクリックすると表示されます) と、従来のデータベース プロジェクト ([その他のプロジェクトの種類]、[データベース]、[データベース プロジェクト] を順にクリックして表示される場所にあり、一般的に、結合されていない .sql スクリプト ファイルのコレクションのみを表します) を混同しないように注意してください。 データベース プロジェクト (.dbproj) は、従来のデータベース プロジェクト (.dbp) の後継プロジェクトで、Visual Studio 2010 で使用できる唯一のデータベース プロジェクトです。以前のバージョンの Visual Studio では、データベース プロジェクト (.dbproj) は、Visual Studio 2005 Team Edition for Database Professionals と Visual Studio Team System 2008 Database Edition のみに含まれていました。

このガイドは、5 つのトピック領域に分かれています。各領域は、プロジェクトで発生するデータベース ライフサイクルに合致したものになっています。

Visual Studio 2010 データベース プロジェクト ガイドのトピック領域は、次のとおりです。

- **ソリューションとプロジェクトの管理**: Visual Studio データベース プロジェクトのソリューションとプロジェクトのファイルを作成して管理する既定のエクスペリエンスです。ソリューションとプロジェクトの管理のトピックでは、少数のメンバーで構成された開発チームや単純なデータベース構成のほとんどの機能要件に対応するのに必要な情報を提供します。ただし、ソリューションとプロジェクトの構造を慎重に考慮すると、エンタープライズ環境の生産性の向上や適切な配置に大きな効果を発揮します。このトピックでは、プロジェクトのセットアップ方法や必要なプロジェクトの種類について説明します。

- **ソース コード管理と構成管理**: ソースコード管理は、アプリケーション ライフサイクルにおいてきわめて重要な部分です。適切に計画しないと、サポート目的で、さまざまなバージョンのコードを管理するのが複雑になる場合があります。同時に使用するバージョンの管理、ソース コード分岐、およびマージは、データベース スキーマを含むアプリケーションの変更を管理するうえで必要不可欠です。このトピックでは、ソースコード管理、分岐、およびマージの推奨事項をデータベース プロジェクトに適用する方法について説明します。

- **プロジェクト システムを使用した外部の変更**: ソースコード管理と宣言を使用してデータベースを開発する方法を遵守するのが第一原則ですが、この規則を破らず、かつ管理されたデータベース モデルの環境から離れずに、変更を組み込むための方法に取り組まなければならない場合があります。このトピックでは、この原理の重要性と代替策が必要かどうかを判断する方法について説明します。

- **ビルドと配置**: ビルドは、実際にプロジェクトを配置する前にだけ行う手順ではありません。ビルドは、プロジェクトの整合性と品質を保つために、配置場所で繰り返し行う手順になる場合もあります。このトピックでは、開発ワークフローにデータベースのビルドを統合する方法、アプリケーションの配置プロセスを向上するさまざまな方法について説明します。具体的には、WiX (Windows Installer XML) を使用した自動インストーラー ベースの操作と手動の検証プロセスを行う方法を紹介します。

- **データベース テストの検証**: Visual Studio の単体テストには、データベース ライフサイクルで自動テストを使用してコードの品質を確保するための優れた方法があります。このセクションでは、この重要なデータベース開発の部分を簡単に作成して開発プロセスに組み込む方法について説明します。

データベース プロジェクトとデータベース アプリケーション プロジェクト (DAC) の違い

Visual Studio 2010 では、データベース プロジェクトと共に、SQL Server データ層アプリケーション (DAC) という新しいプロジェクトの種類が導入されました。この 2 つのプロジェクトの種類には、データベース スキーマをサーバーに配置するという共通の目的がありますが、セマンティクスと処理は異なります。このガイドでは、データベース プロジェクトについてのみ説明します (DAC プロジェクトの機能については取り上げません)。ここでは、シナリオに基づいて適切なプロジェクトを簡単に選べるようにするために、DAC プロジェクトのアプリケーション ライフサイクルを概説し、DAC とデータベース プロジェクトの主な違いについてまとめました。

DAC の一般的なライフサイクルは次のとおりです。

- DAC は、新規に作成および開発するか、既存の DAC パッケージからインポートするか、Visual Studio 2010 または SQL Server Management Studio で既存のデータベースからリバース エンジニアリングします。
- 開発が完了したら、開発者が DAC パッケージを作成します。DAC パッケージは移植性の高い XML ファイルです。このファイルでは、すべてのオブジェクトを定義し、SQL Server のインスタンスが DAC をホストするために満たさなければならない条件を指定しているサーバーの選択ポリシーをサポートします。
- データベース管理者は、SSMS を使用して、SQL Server 2008 R2 を実行している SQL Server のインスタンスまたは SQL Azure に DAC パッケージを配置します。
- DAC パッケージのコピーは、MSDB データベースに格納され、今後参照したり、アーカイブ目的のために使用されます。
- DAC プロジェクトのスキーマで変更が発生し、新しい DAC パッケージが作成されると、パッケージは再度サーバーに配置されます。データが従来のスキーマを使用するデータベースに存在している場合は、データをデータベースの新しいスキーマに移行およびインポートする必要があります。

この 2 つのプロジェクトの種類の主な違いは次のとおりです。

- DAC パッケージは、SQL Server 2008 R2 を実行している SQL Server インスタンスまたは SQL Azure にのみ配置できます。
- DAC では、SQL Server オブジェクトのサブセットしかサポートされていません。詳細については、MSDN の記事「データ層アプリケーションでサポートされている機能[1]」を参照してください。
- DAC パッケージでは、既存のデータベースをアップグレードできないので、必ずスキーマを変更した新しいデータベースを配置します。サーバーに新しいデータベース スキーマを配置するときには、古いデータを新しいデータベースに移行する必要があります。

[1] http://msdn.microsoft.com/ja-jp/library/ee362013.aspx

- DAC プロジェクトには、サーバー プリンシパルのような特定のインスタンス レベルのオブジェクトが直接含まれます。オブジェクトは、一般的に、オブジェクトを使用するアプリケーションの要件に準拠しています。
- DAC プロジェクトでは、SQL Server のインスタンスが DAC をホストするために満たさなければならない条件を指定するサーバー選択ポリシーをサポートしています。
- DAC プロジェクトをサーバーに配置すると、DAC インスタンスが作成されます。このインスタンスは、SQL Server Management Studio (SSMS) の特殊な機能を使用して簡単に制御できます。

一般的な DAC プロジェクトは、小規模なデータベース プロジェクトを対象としています。たとえば、ISV で簡単に使用して、共通の標準スキーマをサーバーに設定できます。データベースが頻繁に変更されず、DAC でサポートされない機能やオブジェクトが不要な場合は、標準化された展開スキーマを使用することをお勧めします。DAC の詳細については、MSDN の記事「データ層アプリケーションの概要[2]」を参照してください。

一般的なシナリオ

このガイドでは、一般的なシナリオにおけるベスト プラクティスを紹介します。シナリオでは、組織に、小規模な IT 組織、専門のデータベース アプリケーション開発チーム、およびデータベース管理者が含まれるものと仮定します。シナリオは、このガイドを組織に適用する方法を理解していただくためのものですが、このシナリオとご使用の環境が違っていても、ガイドの有用性が損なわれるわけではありません。ここでは、主に Adventure Works サンプル データベースを使用します。このデータベースは、Codeplex の SQL Server のサンプル セクション (英語)[3] から入手できます。

- 組織
  - IT チームが会社の Web ベースの販売システムを開発してサポートしている
  - データベース アプリケーション開発専任のチームを雇用している
  - 会社のデータベース インフラストラクチャをサポートするデータベース管理者を雇用している
  - 社内外のシステムを管理およびサポートする運用管理チームを雇用している
- 組織で使用している Visual Studio ソリューション
  - Visual Studio Web プロジェクト
  - Visual Studio データ アクセス プロジェクト
  - Visual Studio データベース サーバー プロジェクト
  - Visual Studio データベース プロジェクト (主なアプリケーション データベース)

[2] http://msdn.microsoft.com/ja-jp/library/ee362011.aspx

[3] http://www.codeplex.com/SqlServerSamples (英語)

    - Visual Studio データベース プロジェクト (参照データベース)
- 組織に異なる Active Directory ドメインの複数の環境 (開発、テスト、および運用) がある
- 組織で同時に 3 つのバージョンのアプリケーションを保持して使用している
    - 開発 (バージョン N + 1): アプリケーションの次のバージョンを含む開発環境
    - テスト (バージョン N): 運用環境に移行するアプリケーションの次のバージョンを含むテスト環境
    - 運用 (バージョン N - 1): 現在リリースされているアプリケーションのバージョンを含む運用環境
- 組織で Microsoft Team Foundation Server を使用してソリューションのソースを管理している
- 組織でイテレーションとバージョンを使用してソース コードを分岐している
- 組織で自動テストを使用してアプリケーションのビルドを確認している (データベース単体テストも含む)
- 組織で Microsoft Team Foundation Server のチーム ビルド機能を使用して、アプリケーションのビルドと配置を自動化している
    - アプリケーションの開発環境がチーム ビルドの継続的インテグレーションで更新されている
    - アプリケーションのテスト環境がスケジュールされたチーム ビルドによって更新されている
    - アプリケーションの運用環境が、より多くのチームと関係者によってビルドが承認された後に、運用チームが実行する制御された配置を通じて更新されている

次の図について考えてみましょう。この図は、上記の架空の組織を表していて、この組織の環境、アクター、およびプロセス フロー間の関係を示しています。

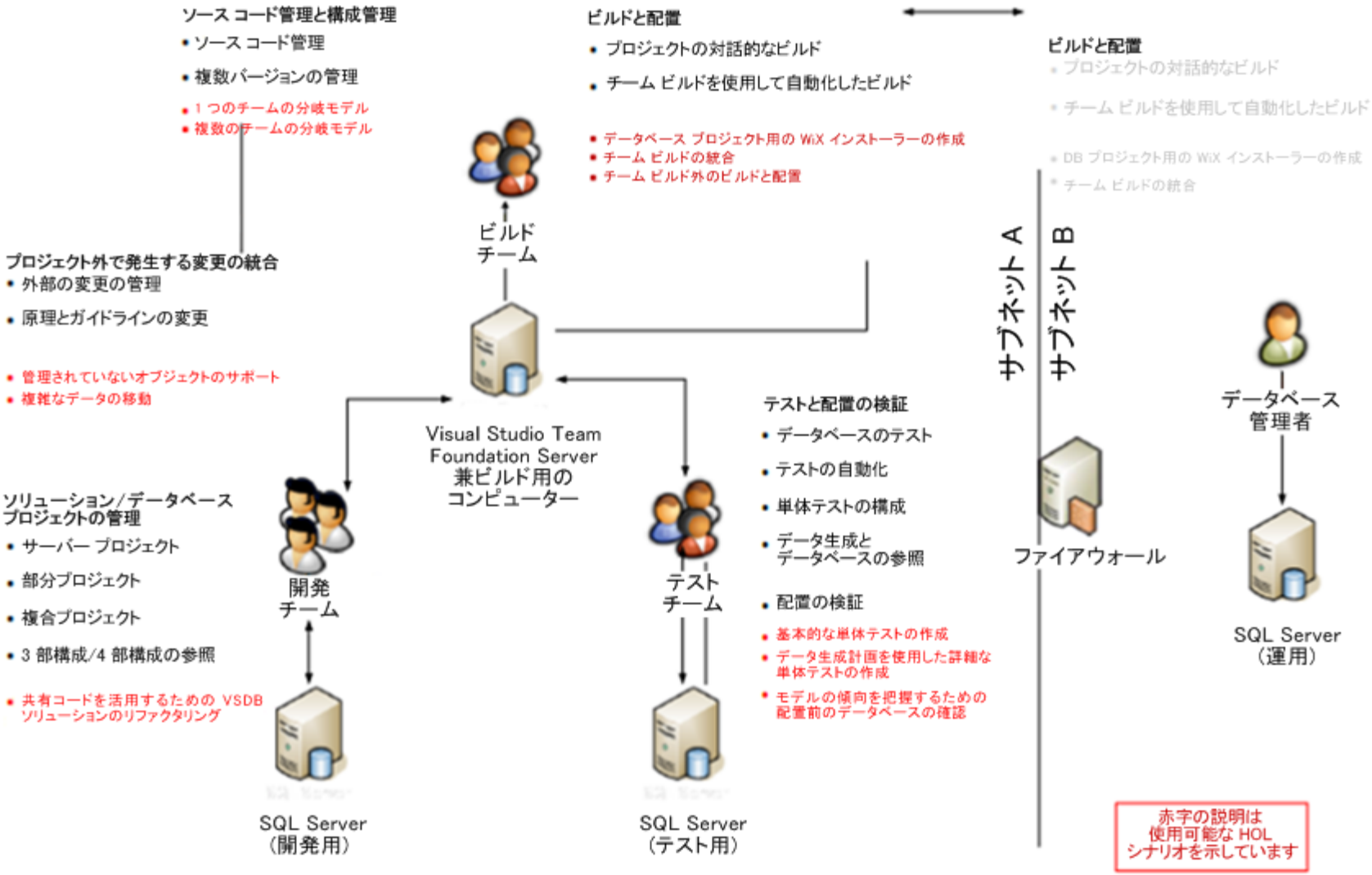


図 1: データベース ライフサイクルの一般的なシナリオ

概要

データベース プロジェクトの主要機能

Visual Studio データベース プロジェクトで提供される機能は 3 つのカテゴリに分類できます。

データベースの変更の管理

- データベース スキーマのバージョン管理: 完全な機能を備えたプロジェクト システムでソースコード管理の統合 (SCC) を使用して、データベース スキーマのオフライン開発の忠実度を最大限に高めます。
- 増分配置: データベース スキーマを複数のデータベースに配置して、ソース コードと異なるスキーマのみを更新します。
- データとスキーマの比較: データベース、データベース プロジェクト、および dbschema (データベース プロジェクトのビルドによる主な出力の成果物) 間で、スキーマとデータを比較します。
- スキーマ リファクタリング: 開発サイクルでデータベース スキーマに繰り返し加えられる、共通する変更の確度を高めます。

データベースの品質の管理

- 依存関係の検証と視覚化: 設計時とビルド時にプロジェクトを検証して、データベース スキーマの一貫性と整合性を確保します。
- 静的コード分析: データベース スキーマ コードで設計、パフォーマンス、および名前付けの問題を特定します。
- 単体テストとデータ生成: アプリケーション開発者が使用するのと同じようなデータベースの単体テストを作成しながら、データ生成機能を使用して、配置されたデータベース スキーマとデータをテストおよび検証します。

データベース開発ライフサイクル (DDLC) とアプリケーション ライフサイクル (ALM) の統合

- 単一の IDE: データベース プロジェクトのプロセスと開発者のエクスペリエンスに一貫性があるので、他の Visual Studio プロジェクトと同じように、データ層の開発者がチーム ベースの開発ライフサイクルに参加できます。
- MSBuild とチーム ビルドを使用したプロジェクト ビルドの統合
- Microsoft Team Foundation Server を使用したソースコード管理の統合

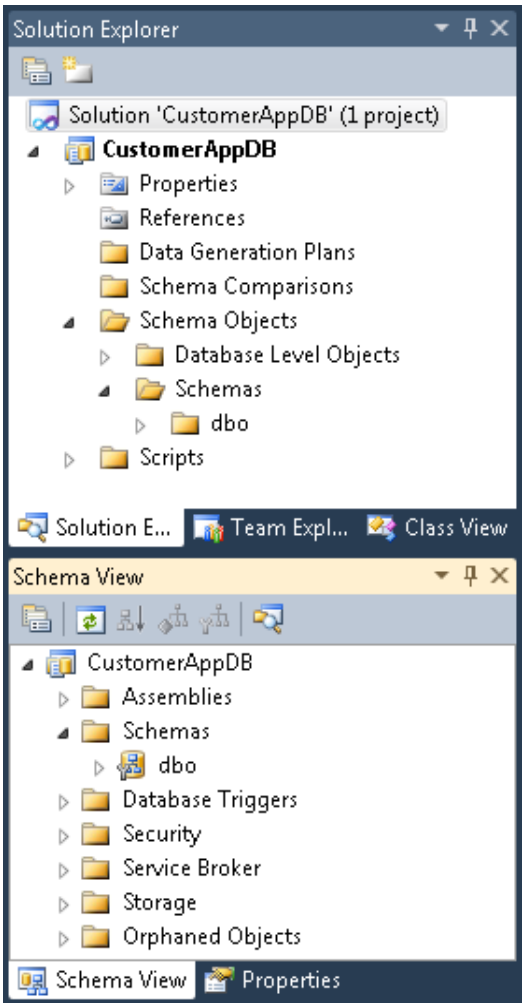


図 2: Visual Studio 2010 でソリューションとスキーマを表示した状態

データベース プロジェクトの概要

Visual Studio 2010 Premium には、アプリケーションとデータベースに加えられた変更を管理する機能が用意されています。データベース プロジェクトでは、次のことが可能になります。

- データベース コードをオフラインで管理できます。
- アプリケーション コードと同じまたは似たような方法でデータベースの成果物を管理できます。
- 共通のソース リポジトリでアプリケーションのソース コードとデータベース コードを同時に管理できます。
- 統合された開発環境ですべてのコードを開発できます。
- ソース コードとターゲット データベース間のデータベースの変更を統合するのにかかる労力が軽減されます。

データベース プロジェクトの機能が使用できるのは

- データベース プロジェクトの中核となるシステムは、Visual Studio 2010 Professional で使用できます。そのため、Visual Studio 2010 Professional を使用している開発者は、Visual Studio 2010 Premium または Visual Studio 2010 Ultimate を使用して作成されたデータベース プロジェクトを読み込み、編集、ビルド、および配置できます。
- Visual Studio 2010 Premium と Visual Studio 2010 Ultimate には、データベースの変更を管理するのに役立つ高度なデータベース ツールが含まれています。

Visual Studio2010 Professional と Visual Studio2010 Premium のデータベース機能

データベース プロジェクトは、Visual Studio 2010 のプロフェッショナル向け開発ツールの 1 つです。データベース プロジェクトとその機能は、Visual Studio の 2 つのエディションで使用できるので、開発チームと配置チームは、ニーズに合わせて適切なエディションを選択して、データベース ライフサイクル全体をとおしてデータベース プロジェクトを使用できます。

各製品で提供される具体的なデータベース機能は、図 3 のとおりです。

図 3: Visual Studio データベース プロジェクトのサポート状況

* Visual Studio Professional の SKU のライセンスでは、既存の単体テスト、静的コード分析のルール、およびデータ生成計画しか実行されない場合があります。これらの機能は、開発者が Visual Studio Premium または Visual Studio Ultimate を使用して実装している必要があります。

これらの機能は、複数のエディションの Visual Studio 2010 で使用できるので、チームは、複数のエディションの Visual Studio 2010 が混在する環境で、データベース プロジェクトを使用してデータベースの変更を管理できます。そのため、Visual Studio Professional しか所有していないチーム メンバーが、データベース プロジェクトをビルドして配置することが可能になります。スキーマ リファクタリングなどの高度なデータベース ツールが必要なチーム メンバーは、Visual Studio 2010 Premium または Visual Studio 2010 Ultimate のライセンスを通じて、これらの機能を使用できます。Visual Studio 2010 では、Visual Studio 2010 Premium と Visual Studio 2010 Ultimate で同じデータベース プロジェクトの機能を使用できますが、他の機能は Visual Studio Ultimate のバージョンでしか使用できません。

組織のニーズとプロジェクトにおける具体的な職務に応じて、担当する作業に適した Visual Studio のエディションを選択できます。

次の表に、Visual Studio 2010 の各エディションで使用できるデータベース プロジェクトの機能とその動作を示します。"実行のみ" となっている箇所は、成果物 (単体テストなど) を変更することはできませんが、実行して出力を得られることを意味します。

機能	Visual Studio 2010 Professional	Visual Studio 2010 Premium/Ultimate
スキーマ比較		✓
データ比較		✓
データ比較	実行のみ	✓
Transact-SQL リファクタリング	実行のみ	✓
Transact-SQL スタティック コード分析	実行のみ	✓
データ生成	実行のみ	✓
Team Foundation Server の ビルド統合	✓	✓
コマンド ラインからの配置 (VSDBCMD)	✓	✓

Transact-SQL のコード作成

データベース プロジェクトのシステムでは、Transact-SQL のコード作成とコード編集のエクスペリエンスが強化されています。データベース プロジェクトでは、IntelliSense とコード スニペットの機能を備えた Transact-SQL エディターが提供されます。また、データベース プロジェクトで Transact-SQL デバッガーを使用することもできます。Transact-SQL のコード作成エクスペリエンスは、SQL Server Management Studio (SSMS) に匹敵するくらい機能が充実しているだけでなく、これらの優れた機能はオンラインとオフラインのデータベース開発で使用できます。

図 4: 統合された IntelliSense により強化されたコード作成エクスペリエンス

データベース プロジェクトの拡張機能

データベース プロジェクトは、拡張可能なデータベース開発エコシステムです。拡張可能なものは次のとおりです。

- データベース スキーマ プロバイダー (DSP)
  - Visual Studio 2008 Team System Database Edition の場合: SQL Server 2000、SQL Server 2005、SQL Server 2008
  - Visual Studio 2010 の場合: SQL Server 2005、SQL Server 2008、SQL Server 2008 R2
  - Visual Studio 2010 では、プロバイダーはパブリックに拡張できます。
    - Quest Software で提供されている Visual Studio 2010 用 Oracle DSP プロバイダー[4] (英語)
- データベース プロジェクトのビルドと配置の拡張機能
  - ビルドと配置の拡張機能によって、外部ツールを開発プロセスにより緊密に統合して、新しい成果物を生成し、配置計画をカスタマイズできます。
  - Visual Studio 2010 では、データベース プロジェクトのビルドと配置をパブリックに拡張できます。
- データベース プロジェクト機能の拡張性
  - コンテキスト ベースの機能の拡張性は、DSP プロバイダーごとにサポートされます。
  - Visual Studio 2010 では、プロジェクト機能をパブリックに拡張できます。
- 機能の拡張性
  - 既存のツールと機能を拡張できます。
  - Visual Studio 2008 Team System Database Edition GDR 以降のバージョンでは、機能をパブリックに拡張できます。

[4] http://toadextensions.com/index.jspa (英語)

図 5: プロジェクトのシステムと機能の拡張

図 6: データベース プロジェクトの拡張機能

拡張機能の詳細については、MSDN の記事「Visual Studio のデータベース機能の拡張[5]」を参照してください。

[5] http://msdn.microsoft.com/ja-jp/library/aa833285.aspx

オフライン スキーマ開発

これまで、多くの組織でデータベースを開発するときには、ソフトウェア開発プロセス以外のさまざまなプロセスが伴っていました。たとえば、多くの場合、データベースの開発時には、運用中の開発データベースに変更を直接加えて、後で参照できるように、その変更をスクリプトまたはドキュメントでキャプチャしようとしたことがあるのではないでしょうか。このデータベースを使用する Web サイトでは、ASP.NET 開発者によって何度か変更が行われ、ソースがなんらかのソース コード制御システムにチェックインされるとします。Web サイトを QA 環境に移行する準備が整ったら、Web サイトはビルドされて XCopy コマンドで QA 環境に配置されます。この際、データベースの変更も移行する必要があるので、データベース管理者または開発者は、追加で作成したスクリプトを使用するか、変更ドキュメントに記載された変更内容に基づいて変更を反映するスクリプトを作成し、QA 環境のデータベースを更新して、すべての変更を反映します。Web サイト アプリケーションの開発と XCopy コマンドによる配置は、データベースを手動で更新するアプローチと大きく異なります。

> アプリケーション開発とデータベース開発における開発と配置の技法の違いは、すべての運用データベースに本質的に備わっているデータと操作状態に起因します。

アプリケーション開発とデータベース開発における開発と配置の技法の違いは、すべての運用データベースに本質的に備わっているデータと操作状態に起因します。この操作状態には、データベースの構成、データベース スキーマ、セキュリティ設定、アクセス許可などが含まれます。データベースの操作状況を変更するには、既存の状態をいくらか把握している必要があるので、変更を行うには追加の作業が必要になる場合がほとんどです。データが存在することによって、データベースの変更プロセスが複雑になります。というのも、アプリケーションの開発時に変更が行われると、多くの場合、データの移行、変換、または再読み込みが行われ、データベースのテーブルや他のデータ スキーマの状態に影響を与えるためです。そのため、開発時の変更プロセスでは、組織にとっての整合性、価値、および実用性を脅かす可能性がある変更から、運用品質のデータと操作状態を保護する必要があります。

データにより課される制限、データベースの状態を管理する一般的な必要性、およびデータベースを変更するために必要な特定のツールによって、多くの場合、組織はデータベース開発プロセスをアプリケーション ライフサイクル管理 (ALM) の戦略に組み込むのに苦労します。このような状況により、アプリケーション開発ワークフロー プロセスの分割と （多くの場合は） 断片化が生じ、結果的に、組織のコラボレーション、効率、および敏捷性の低下を招きます。

この断片化されたプロセスは、一般的に、ソフトウェア構成管理 (SCM) の領域で発生します。データベース開発とアプリケーション開発の SCM の手法を比較した場合、その主な違いは、データベースについて管理しなければならない状態があるかどうかです。この状態を管理するためには、変更を反映するスクリプトを作成する必要があります。このスクリプトでは、データベースで必要なスキーマと状態を特定するだけでなく、配置やリリース時に、データベ

ース (とそのデータ) を既存の状態から新しい状態に移行するのに必要な変更と変換も特定する必要があります。一方、アプリケーション開発では、一般的に既存の状態はわずかしかないので、多くの場合、アプリケーションはリリース時に新しいバージョンに置き換えられます。

> オフライン スキーマ開発では、繰り返しが可能で柔軟性が高いプロセス主体のデータベース開発ライフサイクル (DDLC) を実現できます。

Visual Studio 2010 のデータベース プロジェクトでは、組織の既存の SCM プロセスに簡単に統合できる方法でデータベースの変更を効率的に管理するのに必要なツールが提供されます。Visual Studio 2010 では、データベースのモデル表現に基づいた、ネットワークに接続されていない宣言型のデータベース開発環境が提供され、ソース コードでデータベース、オプション、およびスキーマを定義します。主な成果物は、データベース プロジェクトの Transact-SQL スクリプト (ソース コード) で、他の Visual Studio プロジェクトの成果物と同じ方法で、開発サイクル全体を通して管理およびバージョン管理されます。これは "オフライン スキーマ開発" と呼ばれ、この方法により、繰り返しが可能で柔軟性が高いプロセス主体のデータベース開発ライフサイクル (DDLC) が実現されます。

多くの組織では、バージョン N からバージョン N + 1 にアップグレードできるように、データベースの変更を反映するスクリプトの完全なアーカイブを保持していますが、このようなスクリプトは管理が難しく、実行する必要がある一連の操作と操作を個別に実行する場合を比較すると、後者では余分な手順が必要になります。次の図に、いくつかのリリースでバージョン管理されているテーブルの状態を示します。このテーブルには、データ型が VARCHAR で長さが 25 の description という新しい列がスキーマに追加されていますが、後で長さが 100 に変更されています。

図 7: データベースはデータベース ライフサイクルを通して変更される

従来は、宣言型のデータベース スキーマを使用せずに、バージョン 1 からバージョン 3 への移行するには、次のタスクを実行していました。

- バージョン 1 からバージョン 2 への変更を反映するスクリプトとバージョン 2 からバージョン 3 への変更を反映するスクリプトを提供して、変更を適用します。この方法では、バージョン 1 とバージョン 2 の

いずれかを使用してデータベースを自由にアップグレードできますが、以下のステートメントを使用してスキーマを 2 回変更するオーバーヘッドが発生します。

- *ALTER TABLE dbo.Auction ADD description VARCHAR(25)*
- *ALTER TABLE dbo.Auction ALTER COLUMN description VARCHAR(100)*

- バージョン 1 からバージョン 2、バージョン 2 からバージョン 3、およびバージョン 1 からバージョン 3 への変更を反映するスクリプトを提供します。この方法では、実際の列の変更は 1 つのステートメントで行われます。
  - *ALTER TABLE dbo.Auction ADD description VARCHAR(100)*

  しかし、この方法では、変更前と変更後のバージョンのすべての可能な組み合わせを考慮して、さらに多くの変更を反映するスクリプトを管理しなければならなくなります。

前述の手順と宣言型のデータベース開発を比較すると、後者で保持および処理しなければならないのは、最新の状態のみです。変更は実際の新しいモデルから計算され、データベースに適用されます。

データベース プロジェクトの導入により、一部の開発者は意識の転換に迫られ (古い習慣はなかなか断ち切れません)、少なくともプロセスやワークフローの変更が必要になります。運用データベースを最新バージョンのデータベースと見なすデータベース アプリケーションを開発してきた開発者は、ソース コードでデータベースに変更を加えるというソース コード ベースのアプローチを採用する必要があります。Visual Studio データベース プロジェクトでは、プロジェクトとソース コードがデータベース スキーマの "唯一の真実" で、アプリケーション スタックの他の部分で開発者や組織が既に使用しているような、SCM のワークフローを使用してプロジェクトとソース コードを管理します。データに関しては、当然のことながら運用データベースが "唯一の真実" を保持していることになります。

Visual Studio データベース プロジェクトの導入により、一部の開発者は意識を変える必要に迫られ、少なくともプロセスやワークフローの変更が必要になります。

オフライン スキーマ開発は、実際のデータベースに接続されていない環境でソース コードを使用してデータベースを作成および管理することに重点を置いたデータベース開発アプローチです。ソース コードは、既存のデータベースを変更する際の実装の詳細を定義することなく開発されます。スキーマはソース コードで宣言され、ソース コードは開発ライフサイクル全体を通してバージョン管理されます。また、データベース スキーマは、プロジェクト システムでモデル化され、テーブルやビューなどのデータ オブジェクトの構造と、ストアド プロシージャや関数などのプログラマビリティ オブジェクトが含まれます。

メモ

データベース プロジェクトでは、ほぼすべてのオブジェクトと構文が認識されます。実行時にデータベース エンジンでしか認識されない少数のオブジェクトと構文は除外されています。

Visual Studio データベース プロジェクトでは、次のものを提供することで、オフライン スキーマ開発の環境を実現しています。

- データベース開発を管理するためにデザインされたデータベース プロジェクト (.dbproj)
- スキーマがデータベースのドメイン言語で定義された、宣言型の SQL 構文のサポート
- ソースが、ソースとモデル間をラウンドトリップできる、データベース スキーマのスキーマ モデル表現
- データベースのツールとデザイナーをプログラムで操作できるスキーマ モデルのインターフェイス
- データベースに対して実行することなくソースの整合性を保証できる SQL 構文とスキーマの依存関係の解釈および検証
- ソースとターゲットのデータベースを比較して更新を反映するスクリプトを生成するスキーマの比較エンジン
- 後で実行する配置とスクリプトの生成を可能にするコンパイルされたバージョンのデータベース スキーマ (.dbschema ファイル)
- 構成が異なるさまざまな環境に配置できる .dbschema ファイル

次のシナリオでは、上記の一覧で説明した点を示します。このシナリオでは、Visual Studio データベース プロジェクトを使用せずに、3 つの開発のイテレーション/リリースのサイクルにわたって複数層アプリケーションを追跡します。このシナリオでは、単純なクラスがアプリケーション層を表し、テーブルがデータ層を表します。

バージョン 1 では、アプリケーション開発者が Customer という単純なクラスを定義し、データベース開発者が Customer という単純なテーブルを定義します。

バージョン 2 では、Tom Smith という名前の顧客が複数いる可能性が非常に高いことが判明し、顧客の名前以外に、顧客を一意に特定する方法が必要であると判断します。アプリケーション開発者が Customer クラスに Id メンバーを追加し、データベース開発者が Customer テーブルに Id 列と主キーを追加します。アプリケーション開発者が単純に Customer クラスを更新してチェックインできる一方で、データベース開発者は、複数の ALTER ステートメントを記述して、新しい列と主キーを追加する必要があります。その結果、データベース開発者のコードでは、Customer テーブルの実際の形が表されなくなります。Customer テーブルの実際の形は、最新の開発データベース内に存在し、2 つのスクリプトに及んでいる可能性があります。元のテーブルを作成するスクリプト内にあれ

ば、データベースを更新するスクリプト内にあるものもあります。データベース開発者が変更を加えて、新しいファイルをチェックインします。

メモ

説明を簡潔なものにするため、このシナリオでは、変更をデータベースに追加するために必要なデータの流れは無視します。

バージョン 3 では、顧客をさらに区別する必要が生じ、一緒に事業を行い拡大する顧客を識別するために、顧客にプロパティを追加します。このプロパティを実装するために、アプリケーション開発者は、Customer クラスに IsKeyCustomer という新しいメンバーを追加してチェックインし、データベース開発者は、ファイルを 1 つ追加し、別の ALTER ステートメントを記述して新しい列を Customer テーブルに追加します。データベース開発者は、開発データベースを更新し、スクリプトを変更して、チェックインします。

バージョン 3 の時点では、次のようなプロジェクトの成果物が作成されます。

```
1  using ...
5
6  public class Customer //Added in version 1.0
7  {
8      int id; //Added in version 2.0
9      string fName;
10     string lName;
11     string email;
12     bool isKeyCustomer; //Added in version 3.0
13     DateTime created;
14     DateTime updated;
15     DateTime deleted;
16 }
```

図 8: 複数層のシナリオ - Customer.css

```
--VERSION 1
CREATE SCHEMA [Customer]
GO
CREATE TABLE [Customer].[Customers]
(
    [FName] NVARCHAR(50) NOT NULL,
    [LName] NVARCHAR(50) NOT NULL,
    [Email] NVARCHAR(320) NOT NULL,
    [Created] DATETIME NOT NULL DEFAULT GetDate(),
    [Updated] DATETIME NOT NULL DEFAULT GetDate(),
    [Deleted] DATETIME
);
GO
--VERSION 2
ALTER TABLE [Customer].[Customers]
    ADD [Id] INT NOT NULL IDENTITY(1,1);
GO
ALTER TABLE [Customer].[Customers]
    ADD CONSTRAINT [PK_Customers] PRIMARY KEY
    (
        [Id] ASC
    );
GO
--VERSION 3
ALTER TABLE [Customer].[Customers]
    ADD [IsKeyCustomer] BIT NOT NULL;
GO
```

図 9: 複数層のシナリオ - データベース スキーマの変更

変更を反映するスクリプトに注目してください。このスクリプトは、すべての変更が 1 つの SQL ファイルにまとめられた形で構成されています。一般的に、変更を反映するスクリプトでは、変更を加える前に、予想されるスキーマ条件を検証します。多くの場合、データを含むテーブルに変更を反映するには、データを移動するスクリプトが必要です。すべてのデータベースのバージョンで、スクリプトを何度実行しても結果が同じになるように、ここでは追加の作業を行う必要があります (下図参照)。

```
1  -- VERSION 1.0
2  CREATE SCHEMA [Customer];
3  GO
4  IF OBJECT_ID (N'Customer.Customers', N'U') IS NULL
5  CREATE TABLE [Customer].[Customers]
6  (
7      [FName] NVARCHAR(50) NOT NULL,
8      [LName] NVARCHAR(50) NOT NULL,
9      [Email] NVARCHAR(320) NOT NULL,
10     [Created] DATETIME NOT NULL DEFAULT GetDate(),
11     [Updated] DATETIME NOT NULL DEFAULT GetDate(),
12     [Deleted] DATETIME
13 );
14 GO
15 --VERSION 2.0
16 IF NOT EXISTS (SELECT * FROM sys.columns WHERE name = N'Id'
17     AND object_id = OBJECT_ID (N'Customer.Customers', N'U'))
18 ALTER TABLE [Customer].[Customers]
19     ADD [Id] INT NOT NULL IDENTITY(1,1);
20 GO
21 IF NOT EXISTS (SELECT * FROM sys.key_constraints
22     WHERE name = 'PK_Customers' AND type = 'PK')
23 ALTER TABLE [Customer].[Customers]
24     ADD CONSTRAINT [PK_Customers] PRIMARY KEY
25     (
26         [Id] ASC
27     );
28 GO
29 --VERSION 3.0
30 IF NOT EXISTS (SELECT * FROM sys.columns WHERE name ='IsKeyCustomer'
31     AND object_id =OBJECT_ID (N'Customer.Customers', N'U'))
32 ALTER TABLE [Customer].[Customers]
33     ADD [IsKeyCustomer] BIT NOT NULL;
34 GO
```

図 10: 複数層のシナリオ - 何度実行しても結果が同じになるデータベース スキーマの変更を反映するスクリプト

データベース スキーマの変更を反映するスクリプトの管理には、いくつか問題があります。手動で行うので時間がかかり、エラーが発生しやすく、時間の経過と共に変更バージョンの数が増加するにつれて複雑さが増します。また、開発チームの敏捷性によって変更スクリプトの管理コストが増加するので、あまり拡張可能なプロセスではありません。さらに、オブジェクトを表示または参照するには、運用データベースに対して一連のスクリプトを実行して、データベース スキーマのインスタンスを作成する必要があります。そのため、ソース コードではなく、データ スキーマが "正しいこと" が開発データベースの前提条件となり、依存関係が増えることになります。また、開発データベースと変更を反映するスクリプトで、それぞれ 1 回ずつ、計 2 回の変更を加える必要があります。

今度は、Visual Studio データベース プロジェクトを使用して同じシナリオを実行してみましょう。アプリケーション開発者が行う作業に変更はないので、データベース開発者の作業のみを説明します。

1. バージョン 1 では、Customer という名前の単純なテーブルを定義します。
2. バージョン 2 では、Customer テーブルの定義に Id 列と主キーを追加します。
3. バージョン 3 では、Customer テーブルの定義に IsKeyCustomer 列を追加します。

バージョン 3 の時点では、次のようなデータベース ソースが作成されます。

```
--VERSION 1
CREATE SCHEMA [Customer]
GO
CREATE TABLE [Customer].[Customers]
(
    [Id]    INT NOT NULL CONSTRAINT [PK_Customers] PRIMARY KEY, --Added in VERSION 2
    [FName] NVARCHAR(50) NOT NULL,
    [LName] NVARCHAR(50) NOT NULL,
    [Email] NVARCHAR(320) NOT NULL,
    [IsKeyCustomer] BIT NOT NULL, --Added in VERSION 3
    [Created] DATETIME NOT NULL DEFAULT GetDate(),
    [Updated] DATETIME NOT NULL DEFAULT GetDate(),
    [Deleted] DATETIME
);
GO
```

図 11: 複数層のシナリオ – 最終的な Customer テーブル

データベース開発者は、データベース エンジンで既存のオブジェクトを必要な形に変更する方法ではなく、アプリケーションのバージョンに合わせてオブジェクトの形を定義します。ここで、次のように自問するかもしれません。「既に Customer テーブルがあるデータベースに対して、どのように配置するのだろう」と。このような場合に役立つのが配置エンジンです。前述のように、配置エンジンでは、スキーマのコンパイルされたバージョンと、データベースの配置先を比較します。差分エンジンでは、プロジェクトで配置するバージョンに一致するように、配置先の

スキーマを更新するのに必要なスクリプトを作成します。次の図は、配置エンジンで生成された前述のシナリオの更新を反映するスクリプトです。

空のデータベースに配置すると、配置エンジンでは完全なスクリプトを生成します (以下の抜粋を参照してください)。

```
PRINT N'Creating Customer.Customers...';
GO
SET ANSI_NULLS, QUOTED_IDENTIFIER ON;
GO
CREATE TABLE [Customer].[Customers] (
    [Id]             INT             IDENTITY (1, 1) NOT NULL,
    [FName]          NVARCHAR (50)   NOT NULL,
    [LName]          NVARCHAR (50)   NOT NULL,
    [Email]          NVARCHAR (320)  NOT NULL,
    [IsKeyCustomer]  BIT             NOT NULL,
    [Created]        DATETIME        NOT NULL,
    [Updated]        DATETIME        NOT NULL,
    [Deleted]        DATETIME        NULL
);
GO
SET ANSI_NULLS, QUOTED_IDENTIFIER OFF;
GO
PRINT N'Creating On column: Created...';
GO
ALTER TABLE [Customer].[Customers]
    ADD DEFAULT (getdate()) FOR [Created];
GO
PRINT N'Creating On column: Updated...';
GO
ALTER TABLE [Customer].[Customers]
    ADD DEFAULT (getdate()) FOR [Updated];
GO
PRINT N'Creating Customer.PK_Customers...';
GO
ALTER TABLE [Customer].[Customers]
    ADD CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED ([Id] ASC)
GO
```

図 12: 新しく配置するデータベースのテーブルを作成するスクリプト

たとえば、既存のバージョンのデータベースに配置する場合、バージョン 1.0 が実行されている運用環境では、以下のような変更を反映するスクリプトが生成されます。

```
BEGIN TRANSACTION;
CREATE TABLE [Customer].[tmp_ms_xx_Customers] (
    [Id]             INT            IDENTITY (1, 1) NOT NULL,
    [FName]          NVARCHAR (50)  NOT NULL,
    [LName]          NVARCHAR (50)  NOT NULL,
    [Email]          NVARCHAR (320) NOT NULL,
    [IsKeyCustomer] BIT             NOT NULL,
    [Created]        DATETIME       DEFAULT (getdate()) NOT NULL,
    [Updated]        DATETIME       DEFAULT (getdate()) NOT NULL,
    [Deleted]        DATETIME       NULL
);
ALTER TABLE [Customer].[tmp_ms_xx_Customers]
    ADD CONSTRAINT [tmp_ms_xx_clusteredindex_PK_Customers] PRIMARY KEY CI
IF EXISTS (SELECT TOP 1 1
           FROM   [Customer].[Customers])
    BEGIN
        INSERT INTO [Customer].[tmp_ms_xx_Customers] ([FName], [LName], |
        SELECT [FName],
               [LName],
               [Email],
               [Created],
               [Updated],
               [Deleted]
        FROM   [Customer].[Customers];
    END
DROP TABLE [Customer].[Customers];
EXECUTE sp_rename N'[Customer].[tmp_ms_xx_Customers]', N'Customers';
EXECUTE sp_rename N'[Customer].[tmp_ms_xx_clusteredindex_PK_Customers]',
COMMIT TRANSACTION;
```

図 13: 変更されたデータベースのサンプル テーブル スクリプト

プロジェクトのスキーマを、スキーマのバージョンが異なる複数のデータベースに配置して、プロジェクトで定義されているスキーマを統合できます。次の図に、配置時に発生する現象を示します。

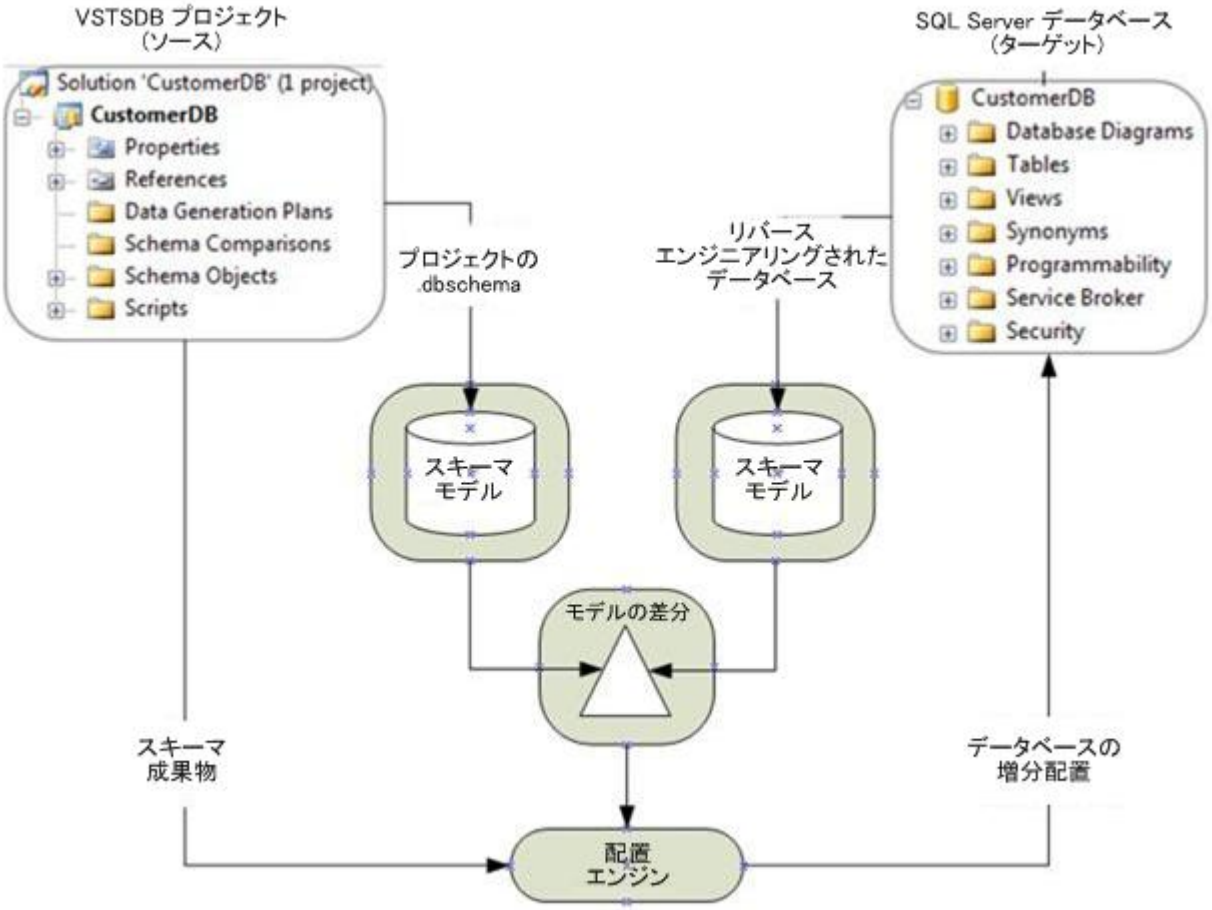


図 14: データベース配置フロー

配置時には、プロジェクトで定義されたスキーマ モデルとターゲット データベースで定義されたスキーマ モデルが比較されます。次に、モデル比較の差分に基づいて、配置エンジンで配置計画が作成されます。

メモ

Visual Studio 2008 GDR から、VSDBCMD.exe という名前のコマンド ライン ツールから、配置エンジンにアクセスできるようになりました。このツールを使用すると、運用環境、Visual Studio をインストールできない環境、またはインストールされている Visual Studio インスタンスが運用環境にアクセスできない環境で、スキーマの比較と配置を簡単に実行できるようになります。

配置計画がターゲット データベースに対して実行されるか、配置計画がスクリプトに配置されます。これは、ターゲット データベースの変更を反映するスクリプトの作成が配置時まで遅延されることを表しています。この仕様により、スキーマを 1 度ビルドするだけで、何度も配置できるようになります。プロジェクトのソースコード管理されるバージョンは、正しいデータベース スキーマを表し、"コンパイルされた" .dbschema ファイルが、特定の時点のスキーマの青写真となります。.dbschema ファイルは移動可能で、アプリケーション リリースのペイロードに含めたり、アプリケーション スタックの他の部分と共に配置できます。

> プロジェクトのソース管理されたバージョンがデータベース スキーマの真実で、"コンパイルされた" .dbschema ファイルが、特定の時点のスキーマの青写真となります。

このことを考慮して、バージョン 1 からバージョン 3 のシナリオを再度見てみると、アップグレードを適用するバージョンは、バージョン 1 と 2 のいずれでも問題ではありません。エンジンでは、現在の正しいソースであるバージョン 3 の状態にするためのスクリプトが作成されます。また、バージョン 3 にアップグレードした場合も、エンジンでは、データベースを再度バージョン 3 からバージョン 2 にダウングレードする配置スクリプトを生成することができます。

推奨事項

開発時と統合時には、配置をテストすることをお勧めします。事前に配置をテストすることで、ターゲット データベースのインスタンスに対して配置エンジンが実行する処理を理解できます。詳細については、テストと配置に関するセクションを参照してください。

スクリプトを配置して、動作を確認してから、テスト環境でスクリプトを実行すると、開発の成果物を簡単に検証できます。また、テスト用のデータベース インスタンスにプロジェクトを配置してテストし、テスト インスタンスを運用データベースまたは開発データベースのスキーマと比較できます。事前にテストすることにより、配置の実行を最終的に自動化するために必要な確信を得られます。開発、テスト、統合、承認、運用前環境、運用など、複数の環境を使用する場合は、実際の運用環境への配置が行われる前に、これらの環境に配置して、スキーマ配置の昇格をシミュレートすることをお勧めします。詳細については、テストと配置の検証に関するセクションで説明します。

ソリューションとプロジェクトの管理

目的

- 主要なエンタープライズ SDLC シナリオを実現する
- 規範的なソリューション/プロジェクトの構造と設定を使用する

概要

Visual Studio データベースのソリューションとプロジェクトのファイルを作成して管理する既定のエクスペリエンスでは、開発者個人の生産性向上を主要な目標とするのであれば、少人数の開発者チームが必要とするほとんどの機能要件に対応できます。ただし、エンタープライズ環境では、アプリケーションの複雑さ、プロジェクトに携わる開発者の人数、および SDLC プロセスの精巧さのすべてが、既定のエクスペリエンスで得られるメリットの低下を招くおそれがあります。

効率的なソリューション構造の特性

ソリューションの構成と管理について綿密な戦略を立てると、ソリューションの潜在的な問題に対処するのに役立つだけでなく、チーム開発の迅速化にも役立ちます。効率的な手法では、次の主な問題に対処する必要があります。

- **機能コンポーネントの分離**: 開発者は、コードを記述するオブジェクトを特定するために、直接体験して得られる下位レベルのコードに関する知識を持っている必要はありません。社内ソリューションの構造は、実用的な検出をサポートし、複数の開発者が個別のコンポーネントで作業を行ったり、個別のコンポーネントを共有したり、使用したりするのに役立つ必要があります。
- **個別の品質基準と開発者の説明責任**: ソリューションでは、完全な回帰テストや統合テストを実行するという厳しい要件を課さず、個々の開発者が機能コンポーネントを個別にビルドして検証できるようにします。これと同時に、継続インテグレーション ビルドと完全なソリューションの配置を、増分変更に対して有効にします。
- **個々の変更の制御と依存関係の管理**: ソリューションの構造は、柔軟なソースコード管理の分岐/統合の戦略、外部コードの依存関係の有効化、または他のプロジェクトと共有するコードの公開に役立つ必要があります。

- **リリースにまたがるソリューションの安定性**: ソリューションの構造は、各リリースで比較的安定している必要があります。新しいビジネス機能や機能コンポーネントの導入により、広範囲に及ぶコードのリファクタリングが発生したり、ビルド定義やリリース計画を再検討したりする必要がないようにします。

論理的なソリューションとプロジェクトの種類

アプリケーション間および開発チーム間で一貫性を確保するため、ソリューションやプロジェクトを作成する時期、目的、および使用方法に関するガイドラインを確立します。Visual Studio データベースですべて明示的に定義されているわけではありませんが、このようなガイドラインを確立する際には、次の論理的なソリューションとプロジェクトの種類について検討します。

Visual Studio データベース プロジェクトのソリューション

このソリューションでは、SSIS、SQLCLR、データを移動するスクリプト、単体テストなど、SQL Server インフラストラクチャを検証するために配置または使用されるすべての関連コンポーネントを含む、アプリケーションのデータベース層を表します。

- **単一のソース分岐**: Visual Studio データベースのソリューションは、単一の単位と見なされるように、特定のソース コード分岐内に全体を配置するようにします。
- **外部依存関係とビルドの依存関係**: データベース プロジェクトの構成要素であるプロジェクトのシーケンスと依存関係は、ここで設定します。DBSCHEMA 外部ファイルの静的なコピーの収集など、ソリューション間の Visual Studio データベースの依存関係はソリューション レベルでも管理する必要があります。
- **環境の選択**: 構成ファイルと構成パラメーターは、ソリューション レベルで設定します。

サーバー プロジェクト

アプリケーションが単一のサーバーと複数のサーバーのどちらを対象に作成されている場合も、この種類のプロジェクトを使用して、サーバーの設定と既定値、およびサーバー レベルのオブジェクトを標準化する必要があります。

- **ソリューション フォルダーにおける統合**: すべてのサーバー プロジェクトは専用のソリューション フォルダーに移動して、簡単に識別したり、コードの開発時にビューに表示されないようにできます。
- **複数サーバーの構成**: 大規模なアプリケーションまたは分散アプリケーションは、複数の SQL Server インスタンスへの配置が必要になる場合があります。サーバー プロジェクトをテンプレートとして使用することで、構成の一貫性を確保できます。

共有ソース プロジェクト

Visual Studio データベースの重要な特徴は、他のプロジェクト用に開発したコードまたはリリース サイクルの早い段階で完全に単独で回帰テストを行ったコードを活用できることです。共有ソース プロジェクトのオブジェクトは、後で、複合プロジェクトとして使用されるか、部分プロジェクト ファイルを通じて使用されます。

- **比較的静的なコード ベース**: この種類のプロジェクトのオブジェクトは、他のプロジェクトのライブラリとして使用されます。そのため、比較的成熟して安定したコードで構成されている必要があります。
- **静的コード分析に関する警告の解決**: このプロジェクトでは、プロジェクトやコードを他の場所で使用する前に、すべてのコード分析に関する警告を解決する必要があります。解決しないと、共有ソース プロジェクトで未解決のエラーや警告が、他のプロジェクトでも重複して発生することになります。
- **ビルドするが運用環境に配置しない**: このプロジェクトのソリューションの設定は、テストを目的として、デバッグ構成でビルドして配置するように設定する必要があります。ただし、この種類のプロジェクトは、リリース構成の運用環境に配置しないでください。別のプロジェクトでコードが参照されると、コードは運用環境に配置されます。
- **個別の単体テスト**: 可能であれば、単体テストを実行し、問題が他のプロジェクトに拡大する前に、このプロジェクトの他の品質基準を評価します。
- **ドメイン データの作成**: ドメイン データを作成するための参照テーブルと配置後に実行するスクリプトは、後で再利用したり、下位プロジェクト間の一貫性を確保したりするために、個別のプロジェクトに統合します。

機能コンポーネントのプロジェクト

基盤となるコンポーネントを個別のサーバー プロジェクトと共有ソース プロジェクトで定義した場合、すべてのコンポーネントを一度に取得するための配置可能なコンテナーが必要です。この種類のプロジェクトは、既定の Visual Studio データベース プロジェクトと関連付けて、機能境界で分離できます。

- **配置可能な個体**: このプロジェクトの出力は運用環境に配置します。
- **プロジェクト間参照の使用**: このプロジェクトでは、確立されたコード ベースを上書きする形で機能または値を追加します。データベース間参照と共有コード ライブラリは、ここで使用されます。
- **アプリケーションの機能的な境界の確認**: 1 つまたは少数の機能コンポーネントのプロジェクトにビジネス機能要件をまとめられるできる場合があります。これは、特定のリリースに回帰の機能を含めるのに役立ちます。

  たとえば、同じデータベースにアクセスする複数のアプリケーションやワークフローが存在するとします。個別のプロジェクトで、それぞれに必要なストアド プロシージャ、ビュー、およびテーブルを実装する場

合、単一のプロジェクトを編集、ビルド、回帰、および配置するときに他のプロジェクトに与えるリスクを抑えられます。

- **テストの機能的な境界**: 複数の開発者が同じコード ベースで作業をしている場合、ソリューション全体よりも小さなレベルで、コードの品質を検証して確認することが重要です。チェックインの前に、コードの増分変更と IDE でのローカル ビルドを簡単に回帰できるように、機能的なテスト ケースの対象範囲は、この種類のプロジェクトに限定します。

  開発者によっては、チェックイン時に必要になる変更は最小限に抑えられる (変更が機能領域に限定される) のが理想的だと考える人もいるでしょう。この種類のプロジェクトの構造を使用することで、ソース コードのマージの必要性が軽減します。また、別の領域で同時に作業をしている他の開発者に影響を与える可能性がある回帰を削減します。

ソリューションとプロジェクトの管理の SDLC シナリオ

ソリューションを論理的なプロジェクトの種類で構成すると、エンタープライズ ソリューションや開発チームに付随するロジスティックな懸案事項に対処しやすい状態にあると言えます。

大規模な開発チーム

通常、開発チームの規模は、Visual Studio データベースのソリューションのサイズとソリューションに含まれるオブジェクトの数の指標になります。適切なソリューションの管理手法を使用すると、ソリューションに含まれるオブジェクトの数が増加する傾向が見られます。上記の論理的なプロジェクトの種類には、開発者の生産性を向上するために、次のような手法が用意されています。

- **作業していないプロジェクトをアンロードする**: この手法では、大規模なソリューションが、小さな共有ソース プロジェクトや機能コンポーネントのプロジェクトに、既に分解されていることを前提としています。ソースコード管理の同期、ソリューションの読み込み時間、Visual Studio データベースのメモリの占有領域、静的コード分析の実行時間、およびビルド/配置の実行時間は、開発者の生産性に悪影響を与えることがあります。
- **一度に実行できる変更の範囲を 1 つのプロジェクトとデータベース スキーマに制限する**: ソリューションの機能を適切に分離すると、複数の開発者がそれぞれ一度に 1 つのプロジェクトと、問題なければ 1 つのデータベースに専念して、簡単に同時進行で作業をすることができます。これは、本来、ソリューションが、(オブジェクト型ではなく) スキーマを重視したオブジェクトの構成用に設定されていることを示しています。
- **ローカルの品質基準と説明責任を適用する**: ビルド時の検証と単体テストでは、チェックイン前に IDE のローカル ビルドが正常に実行される必要があります。ただし、すべての開発者がすべてのコンポーネントを毎回同期してビルドする必要があるのは実用的でないため、意図していない結果を回避するには、影響を受けないプロジェクトをアンロードする必要があります。また、プロジェクト レベルのソース コードのマージで発生する競合、ビルドと環境の設定を軽減することもできます。

共有コードのコントラクトと依存関係

どのリリースでも、ソリューションのある場所でコードが変更されたことが原因で、ソリューション全体の回帰が必要となるのは望ましくありません。つまり、安定して再利用可能なコードは、他のプロジェクト、ソリューション、およびチームで共有できるように分離する必要があります。

共有コンポーネントのコントラクトの公開

他のチームやプロジェクトが依存するコンポーネントを管理する場合は、これらの依存関係の範囲を管理して、今後、このようなコンポーネントを自由に変更できる裁量が必要になります。共有コントラクトの “サブスクライバー” との下位互換性を確保する必要がある場合、重要な変更とのバランスを取るという課題を、重要な考慮事項として覚えておいてください。

実際のところ、共有した成果物は、範囲とバージョンの管理の柔軟性が異なる公開されたコントラクトです。

- **ソリューション間のコントラクトに関する DBSCHEMA ファイル**: 内部パートナーと外部パートナーによるファイルの使用を考慮します。プロジェクト全体 (プライベート オブジェクトとパブリック オブジェクトの両方) を使用する必要があるパートナーもいれば、一部のパートナーに渡す DBSCHEMA ファイルを編集して、すべてのユーザーが使用できるストアド プロシージャへの参照以外を削除する場合もあります。これが、最も分離が進んだ形で共有コントラクトを公開して使用する手法で、ソリューションの構造外で唯一使用可能な手法です。下位互換性は、バージョン管理されたリリースやソース コード分岐と関連付けられます。詳細については、MSDN の記事「他のデータベースを参照するデータベースのチーム開発の開始[6]」を参照してください。

メモ

これは、Visual Studio に同梱されている DBSCHEMA ファイルを通じて、master データベースまたは msdb データベースで外部オブジェクトへの参照を使用するときに活用するパターンと同じです。

- **プロジェクト間のコントラクトに関する部分プロジェクト ファイル**: ライブラリのユーティリティ関数、一般的なインストルメンテーション、監査、ログ記録、共有ドメイン テーブル、データ作成スクリプト、およびレプリケーションの発行者に関する情報を、部分プロジェクト ファイルで定義したマニフェストとして共有する必要があります。部分プロジェクトの詳細については、次の MSDN の記事を参照して下さい。
  - チュートリアル: 部分プロジェクトを使用したデータベース プロジェクトのパーティション分割[7]
  - 方法: 部分データベース プロジェクトをインポートおよびエクスポートする[8]
  - 大規模なデータベースのチーム開発の開始[9]

[6] http://msdn.microsoft.com/ja-jp/library/dd193279.aspx

[7] http://msdn.microsoft.com/ja-jp/library/dd193248.aspx

[8] http://msdn.microsoft.com/ja-jp/library/dd172128.aspx

メモ

複数の部分プロジェクト ファイルの定義または DBSCHEMA ファイルでサブスクライバーが必要としているファイルを共有できます。定義済みのオブジェクトを変更する裁量をサブスクライバーに与えない場合は、このオプションを使用します。

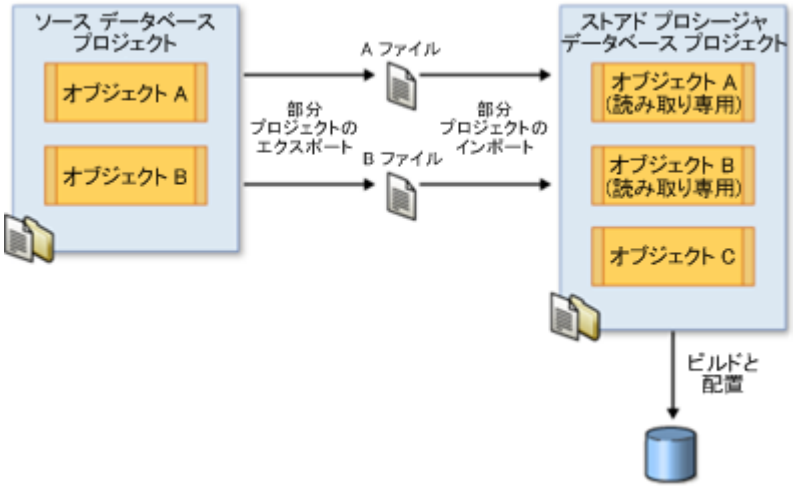


図 15: Visual Studio データベース プロジェクトの部分プロジェクトを使用する

[9] http://msdn.microsoft.com/ja-jp/library/dd193405.aspx#LimitationsOfPartialProjects

- **機能チーム用の複合プロジェクト**: 作業と説明責任を分担するために、個別の機能チームまたは開発者が、機能コンポーネントのプロジェクトを他のプロジェクトでも使用できるように配信します。これは最も簡単に共有コントラクトを公開して使用する方法です。下位互換性はソース コード分岐の戦略と効率と関連があります。詳細については、MSDN の記事「チュートリアル: 複合プロジェクトを使用したデータベース プロジェクトのパーティション分割[10]」と「複合プロジェクトの使用と制限[11]」を参照してください。

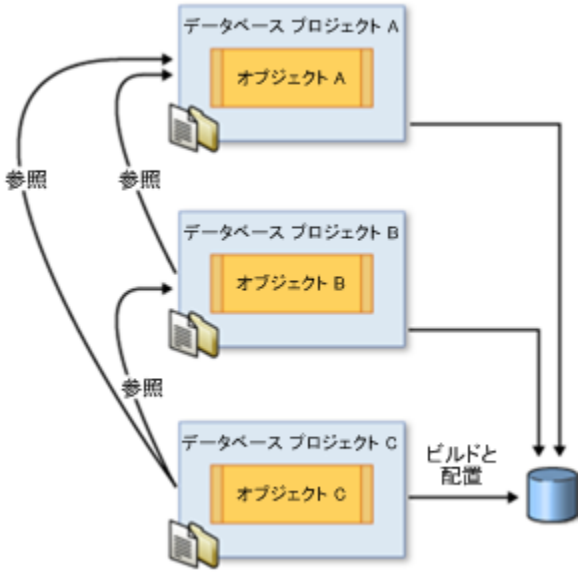


図 16: Visual Studio データベース プロジェクトで複合プロジェクトを使用する

共有コンポーネントのコントラクトの使用

解決すべき重要な問題は、バージョン管理を関連付けるかどうかです。関連する品質基準を満たした修正済みのリリース バージョンとエンド ツー エンドの安定性のいずれかを選ぶ必要があります。共有コードは、DBSCHEMA ファイルやプロジェクト ファイルを参照したり、部分プロジェクト ファイルのマニフェストを利用したりすることで使用します。

- **統合の効率**: 複数のコンポーネントの開発作業を同時に進めている大規模な機能チームまたは複数の機能チームでは、各コンポーネントが安定していることを前提とできる必要があります。使用する依存関係と (独自に開発したコンポーネントの安定性を確保するため) 依存関係を使用するタイミングを管理する場合、修正した DBSCHEMA ファイルを使用します。緊密な統合が必要な場合は、共有コンポーネントまたは機能コンポーネントのプロジェクト レベルで依存関係を使用します。

[10] http://msdn.microsoft.com/ja-jp/library/dd193415.aspx

[11] http://msdn.microsoft.com/ja-jp/library/dd193405.aspx#LimitationsOfCompositeProjects

- **公開するコンポーネントの品質に関する説明責任**: これは本来は発行者が負うべき責任ですが、公開するコンポーネントでは、(いかなる方法を使用した場合も) ビルド エラーおよびスタティック分析の警告/エラーが発生しないことを保証できる必要があります。また、可能であれば、発行者はコードの品質を確認するための単体テストやその他の対策を確保する必要があります。

注

プロジェクト ファイルや部分プロジェクト ファイルを参照するときには、これらのコンポーネントに内在する問題は、参照元のプロジェクトでも問題になります。プロジェクトでエラーや警告を非表示にすることは可能ですが、参照元に影響を与えない場合や、参照元を制御できない場合にのみ非表示にすることをお勧めします。

ソース コード管理と構成管理

目的

- データベース プロジェクトにおける次の処理について説明する
  - ソース コード管理を統合する
  - 複数の環境を対象にした構成を用意する
  - データベース開発をサポートし、管理プロセスを変更する

概要

Visual Studio データベース プロジェクトのオフラインの開発手法が登場するまで、ソース コード管理とさまざまなバージョンのコードの管理は、複雑で必須の処理でした。この手法について説明し、データベース プロジェクトのさまざまなパラダイムについて理解を深めるために、まず、Visual Studio のオフラインのスキーマ開発環境が提供される以前に使用されていた、データベース コードによる管理方法について説明します。その後、データベース プロジェクトのメリットを比較する形で説明します。

Visual Studio データベース プロジェクトが登場する以前のデータベース ソース コードの管理

Visual Studio データベース プロジェクトが登場する以前に最も一般的なデータベース ソース コードを管理する手法は次のとおりでした。

1. 次の例に示すように、さまざまなオブジェクトの SQL スクリプトは、それぞれ個別のオブジェクト フォルダーに配置されます。

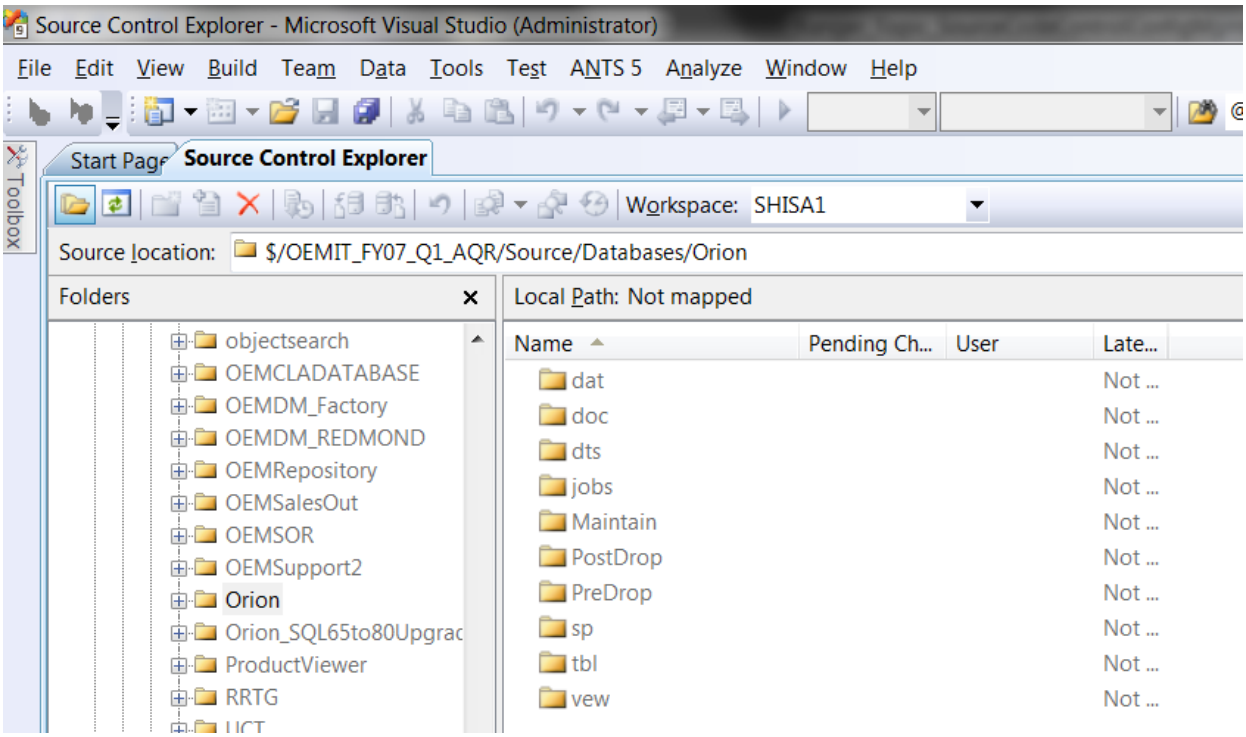


図 17: Visual Studio データベース プロジェクト以前のソースが管理されたデータベース プロジェクトの構造

2. SQL スクリプトは、SQL スクリプトを配置する必要があるターゲット データベースの状態に基づいて作成する必要があります。そのため、ターゲット データベースにストアド プロシージャが既に存在していて、そのストアド プロシージャを編集する必要がある場合は、ストアド プロシージャのスクリプトは、ALTER PROCEDURE ステートメントを使用するか、IF EXISTS (…) DROP PROCEDURE ステートメントを使用してプロシージャを再作成する必要があります。テーブルについても同じ手順が必要になります。ターゲット データベースにテーブルが存在しない場合は、CREATE TABLE スクリプトを作成する必要があります。ターゲット データベースにテーブルが存在する場合は、ALTER TABLE スクリプトを作成する必要があります。

注意

上記のステートメントで対応するのは、スキーマの違いのみです。いくつかのリリースのデータを保持していたり、変更の影響を受けやすいオブジェクトを使用している依存オブジェクトが存在する場合、追加の処理が発生し、スクリプトを作成および管理する必要が生じます。 詳細については、「オフライン スキーマ開発」を参照してください。

SSMS でアドインを使用していない場合、開発者またはデータベース管理者が、データベース サーバーに対して実行するコードを作成し、コンパイルして動作することを確認する必要があります。その後、開発者は、このコードをソース コードに配置する手順を実行し、先ほど説明した変更と移行に関するコードを作成して、ソースコード管理にチェックインする必要があります。

まとめ

データベース ソース コード管理は、ターゲット データベースの状態に左右されるため、このモデルを複数のリリースで利用する場合、ソース コード管理の観点から見ると、ソース コード分岐を使用することはできません。これは、ソース コードの配置 (具体的には、結果として生成されるアセンブリ) を、既存のランタイム ライブラリと置き換えて、実行フォルダーに XCopy できる、一般的なコード プロジェクトで使用される手法とは異なります。データベースを最初から構築しないで済むように、既存のデータベースのデータを保持する必要があるのは、分岐間で移行を行う際に、配置の違いを確認するのに多くの時間を費やすか、またはソースコード管理システムの異なる分岐間で、さまざまなスクリプトの順列を作成したりする必要があることの表れです。

Visual Studio データベース プロジェクトを使用したデータベース ソース コード管理

Visual Studio データベース プロジェクトにより、データベース ソース コードを管理する方法が変化しました。C# コードまたは .NET プログラミングに従事しているデータベース開発者の方は、データベース プロジェクトを使用したデータベース ソース コードの管理方法に関して、これまでの方法と特に変わらないと思われる方もいらっしゃるでしょう。データベース プロジェクトは、Visual Studio ソリューションの C#/.NET プロジェクトで、C# または .NET コードを構築して管理する場合と同じように機能するように設計されています。

> オフラインのスキーマ モデルの開発により、他の .NET コードのように、データベース コードをシームレスに分岐およびマージできる VSTF の機能が使用できるようになりました。

一般的に、データベース プロジェクトには、開発者が、配置先のターゲット データベースの状態に基づいて、データベース コードを作成する必要がないという重要な特徴があります。データベース プロジェクトでは、配置時にスキーマ比較エンジンを使用して配置を行います。これは、実際に、データベース コードをソース コード管理に統合して管理するのに役立ちます。宣言型という Visual Studio データベースの性質を利用すると、他の .NET コードの場合と同様に、データベース コードをシームレスに分岐およびマージできる Visual Studio Team Foundation の機能を使用できます。

複数のアクティブなリリースの同時管理 (分岐)

端的に言うと、ソースコード管理は、特定の分岐のさまざまなバージョンのソース コードを格納することで、確認のために過去のバージョンをロールバックしたり、特定のリリースにロールバックしたりできることです。さまざまなバージョンを管理する大規模なプロジェクト (顧客によって異なるバージョンのソース コードを配置する ISV シナリオなど) では、バージョンが 1 つに限定されます。多くの開発者は、コードを開発する際に、1 つのバージョンのみを使用するという状況に慣れているため、データベース コードにも同じパラダイムが適用されます。このような場合に、ソース コードの分岐とマージが役立ちます。

分岐は、ソース コード管理システムでよく使用される用語です。分岐によってコードが分離されることで、チームは複数のバージョンのコードで同時に作業を行えます。Visual Studio Team Foundation Server で分岐を実現している方法と分岐の使用方法の詳細については、MSDN の記事「Team Foundation Server におけるソース コード管理の使用[12]」と ALM Rangers が提供している分岐に関するガイド「Microsoft Team Foundation Server Branching Guidance[13]」(英語) を参照してください。

Visual Studio プロジェクトとソリューションでソース コードを管理することに慣れていないデータベース開発者とデータベース管理者にとって、分岐は新しい概念でしょう。 Visual Studio データベース プロジェクトでは、ソース コード管理を利用した、統合されたデータベース コード管理と開発エクスペリエンスが提供されます。

さまざまな分岐の戦略については「Microsoft Team Foundation Server Branching Guidance[13]」(英語) を参照してください。

分岐の観点から見ると、ソースコード管理では一般的に次の作業が必要になります。

- データベース スキーマを分岐して、安定したデータベース スキーマまたは基準となるデータベース スキーマ (最後に成功したビルドなど) に基づき、分離された領域を作成します。分岐操作では、ソース分岐 (データベースの取得元) は親分岐と呼ばれ、ターゲット分岐 (分岐により生成され分離されたコピー) は、子分岐と呼ばれます。たとえば、1 つのチームの分岐モデルは次のようになります。

[12] http://msdn.microsoft.com/ja-jp/library/ms364074(VS.80).aspx

[13] http://branchingguidance.codeplex.com/ (英語)

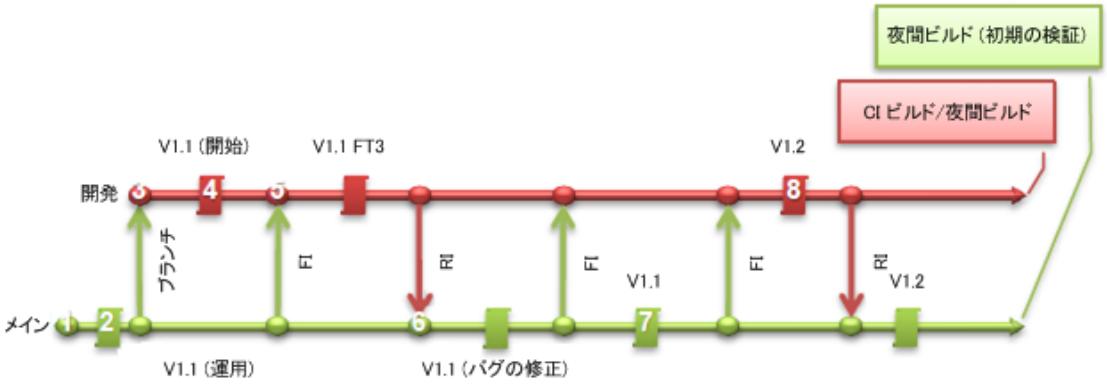


図 18: 単純なチームの分岐モデルの例

- ✓ **開発**: 開発分岐では、機能の開発、新しいリリース (v*N*.*N*) に向けてのバグの修正、重大な変更の統合のいずれであっても、すべての新規開発作業を分離します。この分岐は、新規開発に伴う作業の分離、管理、安定化を実現するために設計されています。
- ✓ **メイン**: メイン分岐は、v*N*.*N* のリリース前に、統合されたすべての機能の最終的な安定化 (QA) を行うために使用します。メイン分岐では、できるだけ安定した状態を維持する必要があります。

- 特定の分岐内のデータベース プロジェクトは他の分岐と分離されているため、開発者はデータベース プロジェクトに変更を加えることができます。
- 開発者は、親子関係にある 2 つの分岐間で、変更のマージを双方向に行えます。親分岐から子分岐へのマージは、順方向統合と呼ばれます。子分岐から親分岐へのマージは、逆方向統合と呼ばれます。マージの関係 (親分岐/子分岐の関係など) が存在しない 2 つの分岐間で、マージを作成することもできます。このようなマージは、ベースレス マージと呼ばれます。ベースレス マージでは、ソース分岐とターゲット分岐間でマージの関係を作成します。
- 上記の ALM Rangers の分岐とマージのガイドラインを参照しながら、ガイドラインのそれぞれの分岐の計画について説明しますが、まずはメイン分岐を取り上げます。メイン分岐は、安定したデータベースを維持することを目的としています。開発分岐は、すべてメイン分岐の子分岐になります。
- メイン分岐は、子分岐である各開発分岐と頻繁に同期することをお勧めします。同期を行うには、メイン分岐から開発子分岐に、通常の順方向統合 (FI) のマージを実行します。
- 一方、ソース分岐のデータベースが安定しており、品質保証 (QA) の品質基準を満たしている場合は、逆方向統合 (RI) のみ実行する (子分岐の開発分岐からメイン分岐にデータベースをマージする) ことをお勧めします。

- メイン分岐を開発分岐と同期した状態を維持することで、メイン分岐で行われた多数の変更を開発子分岐にマージされる “ビッグ バン” マージの問題を回避できます。この問題が原因で、マージ処理を完了する前に、解決しなければならないマージ競合が数多く発生します。

注

データベース プロジェクトを分岐したり、ソースコード管理に初めてデータベース プロジェクトを追加したりするときには、次のファイルのバージョン管理は必要ありません。

- <プロジェクト名>.dbmdl: このファイルはキャッシュされたプロジェクト モデルです。ソースコード管理しないことをお勧めします。IDE でプロジェクトを開いている間は、このファイルに対して読み取り/書き込みのアクセス権が常時必要になります。キャッシュは、チームの開発者ごとに再構築する必要があります。
- <プロジェクト名>.dbproj.user: このファイルは、分離した開発環境の配置設定のような、ユーザー プロジェクトの設定に関するファイルです。開発者は、設定ファイルを各自作成する必要があります。
- Sql フォルダー: データベース プロジェクト配下にあるこのフォルダーには、ビルドの出力が格納されます。このディレクトリ内のファイルでは、ビルド/配置を行うたびに、読み取り/書き込みの処理を実行する必要があります。
- Obj フォルダー: データベース プロジェクト配下にあるこのフォルダーには、ビルドの一時的な出力が格納されます。このディレクトリ内のファイルでは、ビルド/配置を行うたびに、読み取り/書き込みの処理を実行する必要があります。

推奨事項

データベース プロジェクトで参照される .dbschema ファイルが、ソースコード管理で分岐または追加されるように、データベース参照がバージョン管理されるようにします。

データベース参照は、データベース プロジェクトの構成要素ではないため、データベース プロジェクトのソリューション ファイルを開いても、参照しているデータベースの最新の .dbschema ファイルは取得されません。そのため、参照している .dbschema ファイルの最新のバージョンを取得して、ソリューションを頻繁にコンパイルする必要があります。

多くの場合、マージのソース分岐のファイルに変更が加えられると、ターゲット分岐の対応するファイルに変更が自動的にマージされます。マージに含まれるソース分岐とターゲット分岐の同じファイルに変更が加えられると、マージ競合が発生します。変更がファイルの別のセクションに存在する場合、マージ競合はツール (自動マージ) によって自動的に解決されます。マージの実行者が、ターゲット ファイルを上書きする (ソース ファイルの変更を優先する) か、ソース ファイルを無視する (ターゲットのバージョンを維持する) かを把握していることがあります。ただし、マージ競合を解決するときに、手動による操作が必要になる場合があります。このような場合、Team Foundation Server では、マージ競合を解決する 3 とおりの方法を提示するウィンドウを表示し、競合する変更を示します。ユーザーは、ソース ファイル、ターゲット ファイル、およびマージ後のファイルで、データベース コードを確認できます。Visual Studio® 2010 では、競合の解決はモーダルな操作ではありません。この時点で、ユーザーは、ターゲット ファイルとソース ファイルのいずれかまたは両方で特定のデータベース コード行を選択して、最終的な結果にマージできます。

注意

リリース間でデータベース プロジェクトをマージするときは、XML 構造のファイルを確認してください。マージは、ファイル比較ツールを使用して、慎重に実行してください。dbproj、dbschema、.sqlpermission などのファイルは、XML 構造のファイルです。ほとんどの場合、要素はソース ファイルとターゲット ファイルで再構築されるため、XML ファイルのマージは困難な作業になることが多いです。

プロジェクト システムを使用した外部の変更の統合

目的

- プロジェクト システムがソースコード管理と配置にかかわる方法について説明する
- データの複雑な移動シナリオの管理に使用できる技法を概説する
- プロジェクト システムでサポートされない SQL オブジェクトを処理する

概要

Visual Studio データベース プロジェクトは、データベース開発に大きな進化をもたらしました。このプロジェクトに備わっている 3 つの主な機能は、データベース開発を、プログラミング言語を使用する従来の開発に近付けることを目的に設計されています。

- Visual Studio データベース プロジェクトにより、以前には対応する必要があったバージョン管理の問題がなくなります (「オフライン スキーマ開発」セクションを参照してください)。
- Visual Studio データベース プロジェクトのツールには、堅牢な配置エンジンがあります。これを使用すると、データベース プロジェクトを、手動で管理する一連のスクリプト ファイルとしてではなく、パッケージとして配置できます。
- Visual Studio データベース プロジェクトには、単体テスト、リファクタリング、およびスキーマとデータの比較を行うための高度なデータベース ツールも用意されています。

データベース プロジェクトは成熟過程にある製品です。強力な機能は備えていますが、データベース プロジェクトの機能を十分に活用するためには、プロジェクト システムから離れなければならない場合があります。これは、特定の操作をすべてのシナリオに合わせて配置エンジンで自動的に対応できないターゲット データベース システムの制限や、すべての操作やオブジェクトがプロジェクト システム内でモデル化できるわけではないという制限によるものです。

この状況を示すために、ここでは、次のトピックについて説明します。

- 複雑なデータの移動
- サポートされないオブジェクト

外部の変更の管理

外部の変更には、次の 2 種類があります。

- データベース プロジェクトのスキーマ モデル外で発生するデータベースやデータベース スキーマの変更
- データベースやデータベース サーバー プロジェクトに含まれないコード成果物

.dbschema ファイルで表現されるスキーマ モデルは、データベース スキーマの青写真です。このファイルにより、配置中に他のモデルと比較することができます。プロジェクト システムでファイルを細かく分割した成果物として表現されるデータベースとサーバー プロジェクトは、ソース コード管理と統合するためのメカニズムです。

外部の変更を行うことは必ずしも悪いことではありません。データベースやサーバー プロジェクトでサポートされないオブジェクトは、外部で管理する必要があります。また、配置時にデータの移動をより細かく制御する必要がある場合は、プロジェクト システムと格闘し、結果的に、スキーマ モデルやデータベース プロジェクトの外側にあるオブジェクトを操作することになります。

> まず第一に、外部の変更を行うことは必ずしも悪いことではありません。データベースやサーバー プロジェクトで特定のオブジェクトがサポートされない場合は、オブジェクトを外部で管理する必要があります。

これらのシナリオでは、3 つの原理に準拠しようとしています。この原理は、既存のソースコード管理と配置機能をできるだけ多く維持することを目的としています。また、シナリオでは、開発プロセスに自然な形で IDE エクスペリエンスを維持しようとしています。データベース開発でプロジェクト システムを効率的に使用するには、これらの原理を、開発、保守、および配置を簡略化するための大きな柱として使用する必要があります。

原理

- **事前スクリプトと事後スクリプトでオブジェクト管理を行わない**: データベース プロジェクトのシステムでオブジェクトがサポートされていても、配置に関する決定が適切でないと判断する場合は、オブジェクトを事前スクリプトと事後スクリプトで管理しなければならない場合があります。ただし、これは危険な方法で、すべてのオブジェクトを事前スクリプトと事後スクリプトで管理する結果となる可能性があります。オブジェクトをソースコード管理に追加する方法から一度外れてしまうと、プロジェクト システムでは、すべての依存関係を追跡できなくなり、配置時に問題が発生することがあります。配置後スクリプトで列にインデックスを作成する場合は、これに該当します (これはデータベース プロジェクトのシステムでモデル化されません)。同じ名前でモデル化されたインデックスが既に存在する場合は、ビルド プロセスで重複した名前を検出できないので、配置でエラーが発生します。

- **ソースコード管理を優先する**: データベース開発者は、運用データベースをソースコード管理のマスター データベースと見なす傾向があります。これは、アプリケーション ライフサイクル管理に完全に反しており、リファクタリングと単体テストの機能が使用できなくなります。データベース プロジェクトでは、この原則を既定で回避できます。そのためには、プロジェクトを作成するときに、[ターゲット データベースにあってデータベース プロジェクトにないオブジェクトに対して DROP ステートメントを生成する] チェック ボックスをオフにします。

  このチェック ボックスをオフにすると、興味深い現象が生じます。配置時に、プロジェクト システム外にあるオブジェクトが、テスト済みのコードと連動することが保証されなくなります。この現象をテストするための単純な方法は、外部キー リレーションシップを削除または変更してから、外部キーが設定されていた列の名前を変更します。プロジェクト システムから外部キーのファイルの成果物を除外してから、テーブルの定義で列の名前を変更するのが自然な流れです。しかし、[ターゲット データベースにあってデータベース プロジェクトにないオブジェクトに対して DROP ステートメントを生成する] チェック ボックスをオフにすると、配置エンジンでは、テーブルだけが変更されるので、既存の外部キー リレーションシップによって配置でエラーが発生します。その結果、オブジェクトを事前スクリプトと事後スクリプトで管理することになります。

重要

Visual Studio 2010 では、インデックスと制約のみを削除する個別のオプションが導入されました。ただし、モデルが同期された状態を維持して、データベース管理者が定義した操作のインデックスを削除しないようにする必要があります。また、テストに使用するモデルが運用システムに準拠するようにします。

- **IDE エクスペリエンスを維持する**: 配置しないプロジェクト システムに構造の状態を作成したくなるような状況がたくさんあります。これは複雑なデータの移動シナリオで見られます。このアプローチは、"技術

的には" 問題ありませんが、ソースコード管理において多くの悪影響が発生するおそれがあります。これは、ソースコード管理に準拠するために加える必要がある変更が、長期に渡る開発サイクルに基づいているためです。ほとんどの場合、運用環境に配置した後にも、プロジェクト システムに戻ってプロジェクトに変更を加える必要があります。たとえば、データベース スキーマの変更を実際に配置する前に、データをあるバージョンから別のバージョンに移行する必要がある場合などが、これに該当します。やむを得ない場合もありますが、可能な限り、Visual Studio データベース プロジェクトを使用して、配置ツールのデータの移動機能に準拠するようにします。

推奨事項

原理についての説明を読みながら、これらの原理が互いに密接な関係があり、すべての原理が同じステートメントを支持していることに気付いた方もいらっしゃるでしょう。 モデルを主な正しいソースとして使用して、ほぼすべての成果物と変更がモデルで認識されるようにすることで、開発者、テスター、およびデータベース管理者の作業を簡略化して、予期しない問題が発生することを防げます。

複雑なデータの移動

外部の変更の管理における大きな課題はデータです。新しいバージョンが作成されたときに簡単に配置できる .NET アセンブリと比較すると、データベースの課題は、データベースの既存データを維持しながら、スキーマを変更することです。

既定では、[データ損失が発生する場合に増分配置をブロック] チェック ボックスがオンになっているので、通常、この処理は配置時にすべてバックグラウンドで実行され、データ損失を防げますが、これは事前に準備しておくことをお勧めします。

SQL Server の構文と実装の制限により、スキーマの変更を配置する前に、追加の作業を行わなければならない場合もあります。たとえば、変更する属性の値に、既定値ではなく意味のある値を設定して、Null 許容型フィールドを Null 非許容に変更するなどの作業が必要になります。

これを実現する方法の 1 つは、配置前スクリプトをフックして、配置後に Null 非許容に変更する列のデータを収集することです。このスクリプトにより、配置時に制約の規則が実行されるようになります。このトピックの最後で紹介します。

サポートされないオブジェクト

データベース プロジェクトのバージョン履歴でサポートされないオブジェクトの一覧を確認すると、サポートされていないオブジェクトがごくわずかしかないことに気付くでしょう。サーバー プロジェクトの導入により、大きな一歩が踏み出されました。現在では、ほぼすべてのデータベース環境に対応できるようになっています。ですが、まだサポートされていないオブジェクトがいくつかあります。サポートされていないオブジェクトは、次のとおりです。

- リレーショナルな成果物
    - セッションおよびグローバルの TEMPORARY TABLE (ローカルがサポートされます)
    - CREATE RESOURCE POOL
    - CREATE WORKLOAD GROUP
    - CREATE FULL TEXT STOPLIST (旧称、noisewords)
    - SQL ジョブ
    - レプリケーション

- リレーショナルではない成果物
    - SQL Server Reporting Services (SSRS) レポート

- Integration Services (SSIS) パッケージ
- Analysis Services (SSAS) の定義 (MODEL、DMX、MDX)

- サポートされないオブジェクト オプション
  - 変更データ キャプチャ

注

Visual Studio 2010 では、変更追跡がサポートされます。スキーマの変更は配置の検証でブロックされるので、変更データ キャプチャはサポートされません。

  - レプリケートされたテーブル – スキーマの変更は配置の検証でブロックされるので、サポートされません。
  - STORED PROCEDURES の FOR REPLICATION – レプリケーション用に SQL Server で作成されるレプリケーションのストアド プロシージャはサポートされません。
  - CONSTRAINTS の WITH CHECK と WITH NOCHECK

注

制約は配置オプションで処理されます。GDR と GDR2 では、(SQL の既定値により) 必ず WITH CHECK を使用して配置されていました。今後、Visual Studio 2010 では、WITH NO CHECK を使用して配置されるようになり、配置処理の最後にオプションで WITH CHECK を使用できます。

  - 3 部構成の名前による SQL Server にリンクしないサーバー (Oracle、Excel など)
  - TABLE – LOCK ESCALATION
  - インデックス オプション (MAXDOP、DROP_EXISTING、SORT_IN_TEMPDB)

上記のとおり、データベース プロジェクトでモデル化されていないオブジェクトは、それほど多くありませんが、これらのオブジェクトをサポートする必要性に迫られ、いつの間にか運用環境にいずれかのオブジェクトが含まれる状況になることは十分にあります。

CLR アセンブリ

このオブジェクトの種類は、SQL Server 2005 で導入され、中期および長期間にわたって拡張プロシージャの代わりを務めてきましたが、現在 (SQL Server 2010) および次期バージョンの SQL Server では、その重要性が、さらに高まることが予想されます。SQL Server で CLR の成果物を統合する詳細については、MSDN の記事「SQL Server の CLR 統合の概要 (ADO.NET)[14]」を参照してください。SQLCLR アセンブリは、プロジェクトの種類自体が異なるので、データベース プロジェクト外で作成されてビルドされますが、プロジェクト システムでは、SQLCLR アセンブリの存在に関する情報を把握している必要があり、オブジェクトが変更された場合は、多くの依存関係が生じる可能性があります。そのため、スキーマで変更が発生した場合には、完全な配置と増分配置の間も、オブジェクト間のリレーションシップとビルドの順序を追跡する必要があります。データベース プロジェクトの最初のリリースではサポートが制限されていましたが、現時点では、SQLCLR アセンブリは、ネイティブな成果物になっています。SQLCLR アセンブリには、プロジェクト参照を使用する .NET プロジェクトを含められるようになりました。その結果、プロジェクト システムは、発生したすべての変更を簡単に認識して、依存ツリーを優先できるようになりました。

[14] http://msdn.microsoft.com/ja-jp/library/ms254498.aspx

Visual Studio データベース プロジェクトでビルドと配置を自動化する

目的

- データベース プロジェクトが、対話型のローカル ビルドから複数環境にわたる自動化された配置に及ぶさまざまなシナリオで、どのようにビルドおよび配置されるかを解説する
- 現時点でデータベース プロジェクトでサポートされていないシナリオに対処する方法、ビルドと配置に関連する一般的なタスクと潜在的な問題について説明する
- 開発、ビルド、および配置のプロセスを示す

概要

従来、データベース コードのパッケージ化と配置は、複雑な手動によるプロセスで、開発者とデータベース管理者の間で密接な連携が必要でした。

通常、このプロセスでは、特定のターゲット データベースに必要な SQL オブジェクトを特定したり、データベースに "修正プログラムを適用" して SQL コードを最新の状態にするための更新スクリプトを作成したりします。多くの場合、このデータベースの更新スクリプトは、開発者が手動で作成し、変更が悪影響をもたらさないかどうかデータベース管理者が注意深く検証します。

「一般的なガイダンス」で言及した一般的なシナリオのように、複数のテスト環境 (開発、検証テスト、受け入れテスト、パフォーマンス テストなど) を使用する組織で、問題のないビルドが次の環境に昇格されるというプロセスを採用している場合、単一の更新スクリプトでは、あらゆる環境における想定内と想定外の相違点を十分に調整できないため、新しい開発コードを状態が異なるさまざまなデータベースと同期するプロセスがさらに複雑になります。

データベース プロジェクトは、データベースのビルドと配置のワークフローを簡略化するよう徹底的に設計されています。このため、開発者は、既存の状態について考慮することなく、必要なデータベース スキーマの形に集中してコードを記述することが可能です。

データベース プロジェクトの対話的なビルド

Visual Studio データベース プロジェクトは、他の種類のプロジェクトと同じようにビルドできます。C# などの、他の種類のプロジェクトで使用できるビルド オプションはデータベース プロジェクトでも使用可能です。ただし、バックグラウンドで行われる "ビルド" プロセスは、他の特殊なプロジェクトと同様に、Visual Studio データベース プロジェクト固有のものです。

データベース プロジェクトのモデルの検証

Visual Studio Team System 2008 Database Edition GDR R2 以前では、データベース プロジェクトのスキーマ モデルの保存と検証には、デザイン データベースが使用されていました。デザイン データベースは、データベース プロジェクト (.dbproj) を開くときに動的に作成され、プロジェクト システムでは、プロジェクト システムのバックグラウンドで動作しているデザイン データベース (既定では SQL Server 2005 Express) にスキーマ モデルが保存されました。データベース スキーマのバージョン管理が行われるたびに、変更はこの運用データベース インスタンスに対して検証され、すべてのエラーが Visual Studio のエラー一覧に即座にレポートされるようになっていました。たとえば、定義されていない変数を使おうとすると、エラー一覧に即座にエラーが表示されました。 次に、このエラーの例 (Visual Studio 2008 Database Edition) を示します。

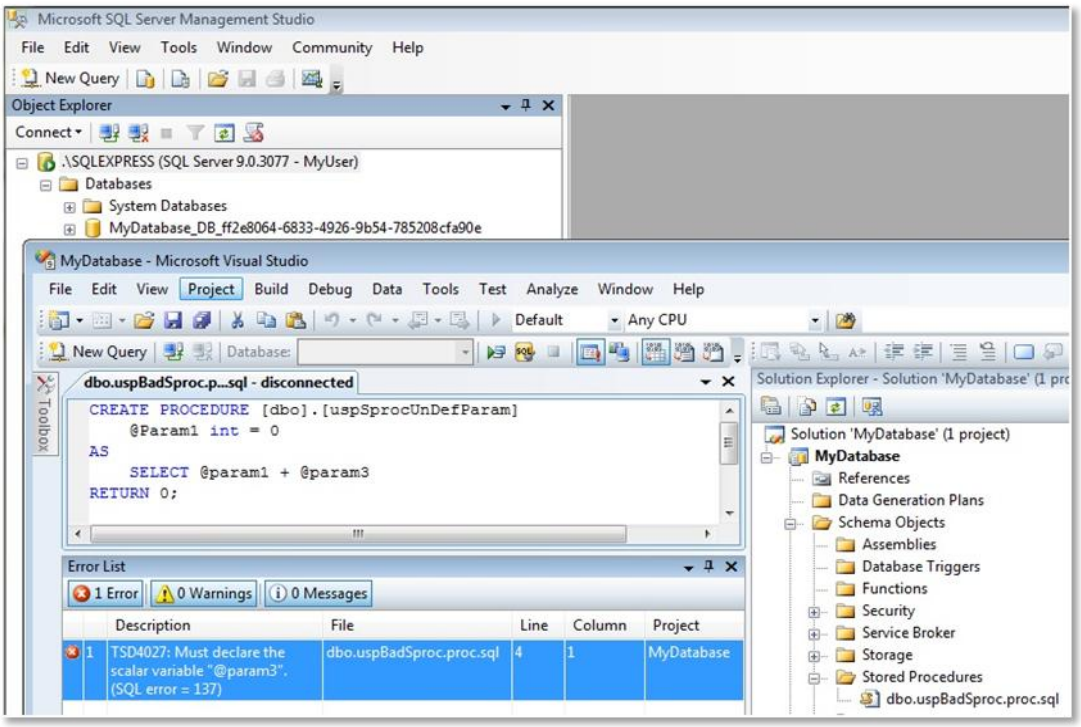


図 19: 検証データベースを使用する従来のプロジェクトの動作

重要

Visual Studio では、プロジェクト システムのデータベース スキーマ モデルの保存と検証に、ライブ デザイン データベースを使用しなくなりました (これは Visual Studio 2008 GDR R2 のリリースからの変更で、Visual Studio 2010 でも引き続き採用されています)。代わりに、データベース スキーマ モデルは SQL Server CE データベースに保存され、検証は、製品の検証エンジンによってデータベース スキーマのメモリ内のモデルに対して実行されます。

これは、製品のアーキテクチャに関する重要な変更で、以前のアーキテクチャでは不可能だった多くのシナリオが可能になりました。データベース スキーマ モデルは、データベース スキーマ プロバイダー (DSP) によって実装され、拡張可能です。このため、SQL Server 環境で作業する場合は SQL100 DSP を使用し、Oracle 環境で作業する場合は ORA11 DSP を使用することになります。このアーキテクチャによって、開発環境にメジャー リリースがなくても DSP モデルを追加および提供することが可能になります。さらに DSP モデルは、単一の .dbschema ファイルにコンパイルしたり、他のプロジェクト機能で使用したり、他の開発チームと共有したりできるようになりました。データ サービス プロバイダーの拡張性については、「Visual Studio のデータベース機能の拡張[15]」を参照してください。

このモデル ベースのアーキテクチャでは、完全な非接続型のデータベース開発環境で作業することになります。プロジェクト システムから運用データベースへの接続は制限され、ターゲット データベース インスタンスに変更をインポートまたは配置したり、ターゲット データベース インスタンスと比較したりするときにのみ接続が可能になります。データベース スキーマの開発時、検証はプロジェクト システムで発生する変更基づいて対話的に実行されます。また、スキーマが変更されていないときにはアイドル状態に基づいて検証が対話的に実行されます。

[15] http://msdn.microsoft.com/ja-jp/library/aa833285.aspx

スキーマの検証は、リソースを集中的に使用するプロセスで、すべてのトップ レベルのオブジェクト、オブジェクト間のすべての参照、およびオブジェクトとオブジェクトを構成する子オブジェクト間のほとんどの依存関係を検証することに重点を置いています。一般的なデザイン時の問題を見つけるための追加の検証もありますが、すべての問題を検出することはできません。

留意事項

スキーマ検証の粒度は、以前のアーキテクチャとは異なります。検証の観点では、GDR とそれ以前のバージョンの間にある最も重要な違いは、プログラマビリティ オブジェクト本体 (関数とストアド プロシージャ) の検証機能がないことです。プログラマビリティ本体において、オブジェクト検証は、開発者の対話的なユーザー エクスペリエンスのパフォーマンスに影響しないよう、最小限に保たれています。ステートメント本体で実行される検証のほとんどは、参照されるすべてのオブジェクトとその子オブジェクトが完全に解決されるようにするためのものです。コードは解析され解釈されますが、すべてが完全に検証されるわけではありません。実際、GDR を使用すると、上記の例のエラーはデザイン時ではなく配置時に検出されます。Visual Studio 2010 では、データベース プロジェクトにより、デザイン時の検証機能が拡張され、宣言されていない変数が検出されるようになりました。

次のシナリオでも、同じ問題による配置のエラーを示しています。

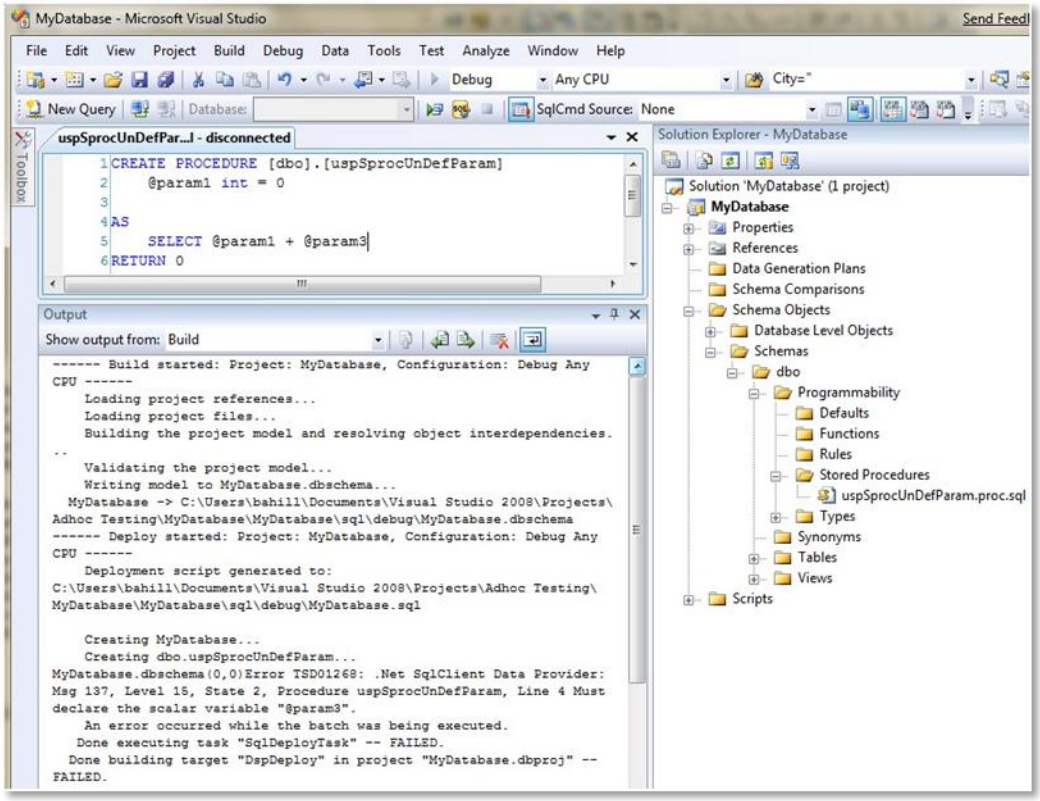


図 20: 特定の検証エラーは配置時にのみに検出される

推奨事項

この変更を踏まえると、配置時にしか検出されないエラーを運用環境への配置前に検出するためには、試験配置を行うことがどれだけ重要かおわかりになると思います。通常、開発中には、データベース スキーマに単体テストを実装するたびに試験配置を何度も行います。

サンドボックス データベースに配置すると、他の環境への配置を行ううえで確信を持てることにもつながります。また、統合中には、多くの環境でサンドボックス データベースへの配置が実行されます。たとえば、統合データベースは、1 日の終わり、または継続的インテグレーションを使用しているときはチェックインによってビルドと配置が開始された後に、作成または更新できます。また、必要に応じて、デザイン時に定義されていない変数を確認するために、カスタムの静的コード分析規則も実装することもできます。重要なのは、早期に配置して、配置を頻繁に検証することです。

サンドボックス データベースに配置すると、他の環境への配置を行ううえで確信を持てることにもつながります。重要なのは、早期に配置して、配置を頻繁に検証することです。

データベース プロジェクトのモデルのコンパイル

データベース プロジェクトは、Visual Studio で [ビルド] メニューの [<プロジェクト名> のビルド] をクリックするだけでコンパイルできます。

データベース プロジェクトをコンパイルすると、データベース モデルが完全に検証およびシリアル化され、.dbschema が生成されます。

コンパイルのフローは次のとおりです。

1) データベース プロジェクトがメモリに読み込まれる (ビルド出力: "プロジェクト ファイルを読み込んでいます")
2) データベース プロジェクト モデルがビルドされる (ビルド出力: "プロジェクト モデルをビルドし、オブジェクトの依存関係を解決しています")
    a. オフラインのスキーマ モデルがキャッシュ (存在している場合) から再作成されるか、プロジェクトの .sql ファイルから構築される
    b. スキーマ モデルが解決され、すべての依存関係が確認される
3) データベース プロジェクトがビルドされる (ビルド出力: "プロジェクト モデルを検証しています")
    a. データベースプロジェクトがビルドされる (ビルド出力:"プロジェクト モデルを検証しています")
    b. プロジェクト設定が処理され、出力パスにコピーされる
    c. 参照されるスキーマ モデルが出力パスにコピーされる
4) .dbschema が生成される (ビルド出力: "<プロジェクト名> にモデルを書き込んでいます")
    a. スキーマ モデルがシリアル化され、.dbschema ファイルにパッケージ化される
    b. 配置マニフェストが作成される
    c. 静的コード分析がビルドで有効になっている場合、分析が実行される。静的コード分析が手動で実行されている場合、分析によってビルド (手順 1. ～ 4.) が開始されることに注意してください 。

データベース プロジェクトをビルドすると、その出力で多くの成果物が作成されます。作成される成果物とその内容の概要は、次のとおりです。

- **.dbschema ファイル**: スキーマ モデルのシリアル化された XML 表現で、データベースの青写真を表します。
- **.deploymanifest ファイル**: プロジェクト レベルの構成情報をすべて含むマニフェストで、データベース プロジェクト ファイル (.dbproj) を使用せずに配置できるようにするために必要です。このマニフェストは、MSBuild プロジェクト ファイルと構造が酷似している XML ファイルです。

- **配置前および配置後スクリプト**: これらの別個のファイルは、ターゲットを作成または更新するスクリプト (配置スクリプト) の前か後に実行されるスクリプトです。使用できる配置前スクリプトと配置後スクリプトは、それぞれ 1 つだけですが、これらのスクリプトに他のスクリプトを含めることは可能です。

注

データベース プロジェクトのみをビルドする場合、ターゲット データベースとの比較は実行されません。このため、一度データベース プロジェクトをビルドしたら、何度でも必要なだけ配置できます。配置時には、差分スクリプトか更新スクリプトが作成され、スクリプトは各ターゲット固有のものになります。

データベース プロジェクトのプロパティ

ビルド プロセスの動作は、データベース プロジェクトのプロパティで定義されます。出力ディレクトリ (OutDir) など、他の種類の Visual Studio プロジェクトと共通しているプロパティもあれば、データベース プロジェクト固有のプロパティもあります。

プロジェクト設定は、データベース プロジェクトのすべてのビルド構成に共通しています。プロジェクト設定の概要とプロジェクト設定がビルドに及ぼす影響の詳細については、「データベース プロジェクト設定の概要[16]」を参照してください。

配置前/配置後イベントの追加

Visual Studio では、プロジェクトのプロパティ ページにある [ビルド イベント] タブを使用して、ビルド前およびビルド後のイベントを比較的簡単にフックできます。これを行うのは、データベース プロジェクトのビルド前またはビルド後に操作を実行するためで、次のようなシナリオで便利です。

- 確実に同期するために、プロジェクト間でアセット (.sql ファイル) をコピーする
- アプリケーション モデルを更新するために別個のツールを実行する

ですが、ビルドの概念は配置とは異なるため、多くの場合、データベース オブジェクトを使用するときには、配置時になんらかの処理を実行する必要があります。そのため、配置前および配置後のイベントをフックする方法が必要です。この方法は、次のようなシナリオに役立ちます。

- データベースのメンテナンスを行うために SQL エージェント ジョブを開始する

[16] http://msdn.microsoft.com/ja-jp/library/aa833291.aspx

- ETL プロセスを開始するために SSIS パッケージを開始する

プロジェクトに配置前イベントと配置後イベントを追加する方法

配置イベントは、プロジェクトのプロパティ ページでは直接公開されませんが、標準的な MSBuild フレームワークで配置イベントが定義されているので、プロジェクト ファイルを手動で更新してフックすることが可能です。

用語の解説

配置前/配置後スクリプトと配置前/配置後イベントを混同しないようにしてください。配置前/配置後スクリプトは、データベース プロジェクトのネイティブのプロジェクト システムに統合されているのに対し、配置前/配置後イベントは、一般的なプロジェクト エコシステムの構成要素です。これらは組み合わせて使用することが可能で、相互に排他的なものではありません。

この手順は、次のように簡単でわかりやすいものです。

1. ソリューション エクスプローラーでプロジェクト ノードを右クリックして [プロジェクトのアンロード] クリックし、プロジェクトをアンロードします。

推奨事項

プロジェクト ファイルは XML ベースで誤って破損しやすいため、プロジェクト ファイルの編集に慣れていない場合は、ファイル システムでファイルのコピーを作成し、作業ファイルのコピーを保険として確保することをお勧めします。

2. 手順 1. の操作により、ソリューション エクスプローラー内のプロジェクト ノードが半透明な状態になり、"(利用不可)" とマークされます。このプロジェクト ノードをもう一度右クリックして、[編集 <プロジェクト ファイル名>.dbproj] をクリックします。これにより、XML エディターにプロジェクト ファイルが読み込まれます。

3. 今度は PreDeployEvent と PostDeployEvent という Name 属性の Target 要素を追加します。

```
1:  <Target Name="PreDeployEvent">
2:      <Message Importance="high" Text="Pre deployment event"/>
3:  </Target>
4:
5:  <Target Name="PostDeployEvent" >
6:      <Message Importance="high" Text="Post deployment event"/>
7:  </Target>
```

図 21: 配置イベントの Target 要素を追加する

4. 変更したら、プロジェクト ファイルを保存して閉じます。その後、再度プロジェクト ノードを右クリックして [プロジェクトの再読み込み] をクリックします。エディターが開いたままになっている場合は、保存して閉じるかどうかを確認するメッセージが表示されます。保存して閉じることに同意すると、プロジェクトがリロードします。

5. これで、プロジェクトを配置して、配置前/配置後イベントが実行されるかどうかをテストできます。すべて問題なく機能している場合は、次のように出力ウィンドウの 9 行目と 14 行目に配置前と配置後のイベント メッセージが表示されます。

```
1:  ------ Build started: Project: nw, Configuration: Debug Any CPU ------
2:      Loading project references...
3:      Loading project files...
4:      Building the project model and resolving object interdependencies...
5:      Validating the project model...
6:      Writing model to nw.dbschema...
7:    nw -> D:\demo\nw\sql\debug\nw.dbschema
8:  ------ Deploy started: Project: nw, Configuration: Debug Any CPU ------
9:    Pre deployment event
10:     Deployment script generated to:
11: D:\demo\nw\sql\debug\nw.sql
12:
13:     The deployment script was generated, but was not deployed. You can change
14:   Post deployment event
15: ========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
16: ========== Deploy: 1 succeeded, 0 failed, 0 skipped ==========
```

図 22: 配置の出力メッセージの例

6. これで、プロジェクトの配置前/配置後イベントにカスタム動作をアタッチすることができます。

注

これらのイベントを作成するときには、MSBuild によってイベントに依存関係チェーンが作成されることを留意する必要があります。その結果として、配置は配置前イベントが正常に完了している状態に依存します。同様に、配置後イベントも、正常に配置が完了している状態に依存します。

関連ファイルをビルドせずにプロジェクトに統合する

シナリオによっては、ビルド プロセスに含めずに、SQL ファイルを作成したり、インポートしなければならないことがあります。この状態を実現するために必要なことは、"ビルド アクション" オブジェクト プロパティを [ビルド内にありません] に設定するだけです。この設定により、ビルド プロセスから SQL ファイルが除外され (ビルド エラーや警告が生成されません)、結果として生成される .dbschema から SQL ファイルが除外されます。つまり、SQL ファイルのコンテンツは配置されません。

また、新しいデータベース プロジェクトにデータベースがインポートされるときに、警告やエラーが生成されることがあります。次に、一般的な警告とエラーを修復する方法についてのアドバイスを示します。

エラー/警告	修復方法
コードでコンピューターや他のアカウント/ログインを参照している	配置後スクリプトで、ドメインまたはコンピューター固有のログインを処理する必要があります。複数の環境に配置する場合は、特定の環境に固有のログインを削除するか、ユーザーの作成に WITHOUT LOGIN を使用する必要があります。
無効なユーザーにアクセス許可が与えられている	Database.sqlpermissions を更新して無効なユーザーを修正します。 さらに変更が必要な場合は、refactoring を使用して、アクセス許可の状態が既存のユーザーと同期されるようにします。
複数のデータベースまたはサーバーを使用するビュー、ストアド プロシージャ、または関数がビルドできない (データベース A が、同じサーバーか異なるサーバーにあるデータベース B のオブジェクトを使用している)	参照データベースまたは .DBschema ファイルに、データベース プロジェクトへの参照を追加します。 .DBSchema ファイルを参照する場合は、最新のスキーマの変更を含めるようにします。最新のスキーマの変更を含めないと、参照オブジェクトが不足している場合にビルドが失敗します。 また、データベース参照では、大文字と小文字が区別されます。
SQL コマンド/ファイルは、SQL Server では適切にコンパイルされるが、Visual Studio ではコンパイルできない	これは一般的なケースではありませんが、既存のデータベース (XML や XQuery など) をリバース エンジニアリングしているときに起こることがあります。トレースを使用して、問題となっている SQL ステートメントを特定します。必要に応じて、式を標準化または置換します。操作のトレースは、レジストリ キーを追加することで実行されます。レジストリ キーを追加する方法の詳細については、「付録」セクションの「データベース プロジェクトのトレースの有効化」を参照し

	てください。
3 部構成の名前の解決に関する問題 TSD4001: オブジェクト名が無効です。データベース参照は未解決です。	データベース プロジェクトでは、自己参照型の 3 部構成の名前をサポートしませんが、変数とリテラルを使用して外部データベースへの参照を解決することはサポートしています。[17]

[17] http://blogs.msdn.com/bahill/archive/2009/08/26/using-self-referencing-or-local-3-part-names.aspx (英語)

チーム ビルドによるデータベース プロジェクトのビルドの自動化

開発速度が上がるにつれ、高品質の製品をより早く提供することが求められるようになりました。これはつまり、.NET コードと同様に、データベース コードのビルドと配置にアジャイル開発の原理を適用する方法を考え始める必要があるということです。開発速度が上がっても、新たなニーズを満たすために、その分だけ重労働になるは本意ではありません。そこで、ビルド自動化が登場しました。

チーム ビルドは Team Foundation Server のコンポーネントです。開発チームでは、このコンポーネントを使用して、アプリケーションのビルド方法を制御したり、アプリケーションを自動化したりすることができます。チーム ビルドでは、5 とおりの方法でビルドを開始できます。ビルドの開始は、トリガーと呼ばれることもあります。チーム ビルドのトリガーは次のとおりです。

- **手動**: ビルドを手動で開始する必要があります。
- **継続的インテグレーション**: ビルドがチェックインのたびに開始されます。詳細については、「Setting Up Continuous Integration with Team Build[18]」(英語) を参照してください。
- **ビルドのロール**: ビルドは、チェックインが行われ、数分経ってから開始されます。詳細については、「Setting Up Continuous Integration with Team Build[19]」(英語) を参照してください。
- **ゲート チェックイン**: ビルドは、正常に完了した場合にのみソース コード管理にチェックインされます。詳細については、「チェックインの前に変更を検証するためにビルドを定義する[20]」を参照してください。
- **スケジュール**: ビルドは特定の時間に開始されます。チェックインが行われていなくても、ビルドが実行されるかどうか判断するオプションもあります。詳細については、「How To: Set Up a Scheduled Build in Visual Studio Team Foundation Server[21]」(英語) を参照してください。

データベース プロジェクトのビルドを開始するためには、どのオプションを実装すればよいのでしょうか。

多くのプロジェクトでは、"継続的インテグレーション ビルド" と "スケジュールされたビルド" の両方が構成されます。これ以降のセクションでは、この 2 つのビルドについて説明して、データベース コードの開発に役立てるうえで、両者が重要である理由を見ていきます。

[18] http://msdn.microsoft.com/en-us/library/bb668957.aspx (英語)

[19] http://msdn.microsoft.com/en-us/library/bb668957.aspx (英語)

[20] http://msdn.microsoft.com/ja-jp/library/dd787631.aspx

[21] http://msdn.microsoft.com/en-us/library/bb668975.aspx (英語)

継続的インテグレーションを使用する場合

継続的インテグレーション (CI) を使用する場合について説明する前に、CI について簡単にまとめましょう。CI は、既存のコード リポジトリに、コードをチェックインするたびに、個別の変更を統合、ビルド、およびテストするという手法です。

継続的インテグレーション ビルドは、実行する数が多いほど良いとされています。

CI の長所には、次のようなものがあります。

- ビルド ブレークを検出できる
- 統合と配置に関する問題を早期に検出できる
- 特定のテストを実行できる
- コードの昇格が自動化される
- 配置に関する問題を特定できる
- 重複する作業を自動化できる
- 関係者がプロジェクトの進行状況を把握しやすい
- コード メトリックスのレポートを自動化できる

継続的インテグレーション ビルドの概念に関する詳細については、「Continuous Integration[22]」(英語) を参照してください。

CI をデータベース プロジェクトで使用する理由

端的に言うと、CI をデータベース プロジェクトの一部として使用する理由は、上に並べたような長所を活用するためです。ですが、データベース チームが CI を実装するのは簡単なことではありません。データベース チームは、CI をアプリケーション コードに統合することには対応できましたが、データベース コンポーネントを CI プロセスに統合することには今も苦労しています。

データベースは "ステートフルである" という点で問題を提起します。CI は、ステートレスなアプリケーションではうまく動作します。というのも、ステートレスなアプリケーションの場合、統合環境またはステージング環境で、以前のバージョンのアプリケーションが、それぞれの新しい CI ビルドと簡単に置き換えられるからです。ですが、多くの場合、データベースでは、既存のデータを維持するため、現在のバージョンからアップグレードする必要があり

[22] http://martinfowler.com/articles/continuousIntegration.html (英語)

ます。このため、アプリケーションのデータベースの新しい CI ビルドを作成するたびに、各バージョンに必要な変更が含まれるよう継続的にデータベースを更新する必要が生じます。

データベース プロジェクトで 1 日の終わりに CI を使用すると、正常なチェックイン、ビルド、および開発サイクルが行われた後に、統合環境とソース コード レポジトリの状態が同期できます。

継続的インテグレーションの詳細については、MSDN ライブラリの記事「How To: Set Up a Continuous Integration Build in Visual Studio Team Foundation Server"[23]」(英語) を参照してください。

スケジュールされたビルドを使用する場合

現在、開発チームで一般的に作成されているもう 1 つの種類のビルドは、スケジュールされたビルドです。スケジュールされたビルドとは、スケジュールに基づいて、将来の特定の時間に開始されるビルドのことです。通常、スケジュールされたビルドは、プロジェクトであまり作業が行われていないときに実行されるように定義されます (通常は毎晩実行するように定義されます)。このため、スケジュールされたビルドは "夜間ビルド" と呼ばれることがあります。

CI ビルドとは対照的に、スケジュールされたビルドは、運用環境に配置されるものをより忠実に模倣します。夜間ビルドは、次のいずれかの処理を行うために構成されます。

- 運用環境にリリースされていないデータベース プロジェクトを削除して作成し直す
- 運用データベースのコピーをアップグレードする

2 種類のビルドを使用する理由

スケジュールされたビルドと CI ビルドの実行時間と継続時間には重要な違いがあります。通常 CI ビルドは、統合環境を継続的に更新する中で、統合に関する問題をすばやく特定できるようにするための、簡単な検証を行うように定義されます。その一方で、夜間ビルドは、より長く実行されるようにデザインされており、包括的な機能統合テストを追加で実行することや、実際の値を設定することによって、運用環境のアップグレードにかかる時間が変わってきます。ですが、夜間ビルドは、問題が特定されたときに開発チームが問題をすばやく修正できるくらいの頻度で実行する必要があることには変わりありません。

[23] http://msdn.microsoft.com/en-us/library/bb668971.aspx (英語)

配置オプション

データベース プロジェクトの配置方法についてのオプションを構成するには、データベース プロジェクト[24] のプロパティを操作する必要があります。このためには、プロジェクトのプロパティの [配置] タブを使用します。

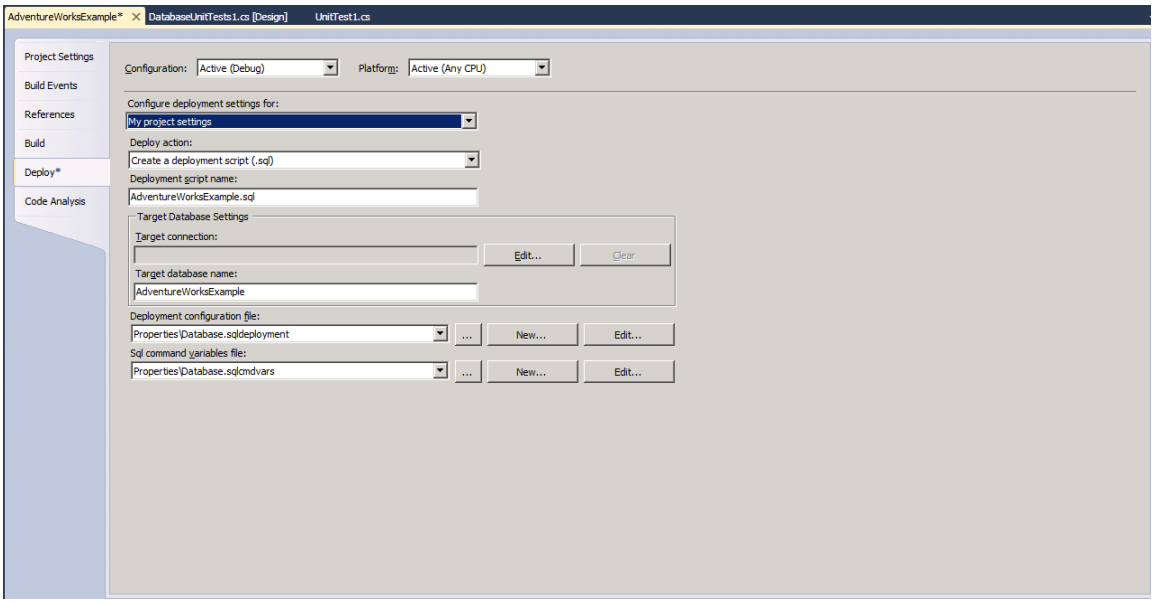


図 23: データベース プロジェクトの配置オプションの構成

[配置] タブでは、次のオプションが強調表示されています。

- **配置設定の構成**[25]: 環境ごとに異なる設定を構成できます。使用できるオプションは次のとおりです。
  - **マイ プロジェクトの設定**: 設定はチーム全体と共有されます。
  - **分離開発環境**[26]: 設定は個人の開発者のみのもので、その他のチーム メンバーとは共有されません。
- **配置動作**: 配置動作が呼び出されたときに発生する処理の詳細を設定します。使用できるオプションは次のとおりです。
  - **配置スクリプト (.sql) を作成します**: 配置 SQL ファイルが配置手順として作成され、それ以外の処理は行われません。
  - **配置スクリプト (.sql) を作成してデータベースに配置します**: SQL スクリプトを作成し、配置構成ファイルで定義されている構成設定に基づいて、スクリプトをデータベースに配置します。

[24] http://msdn.microsoft.com/ja-jp/library/aa833291.aspx

[25] http://msdn.microsoft.com/ja-jp/library/dd193254.aspx

[26] http://msdn.microsoft.com/ja-jp/library/dd193409.aspx

- **配置構成ファイル**: データベースを配置するときに使用される構成ファイルの詳細を設定します。単体テストでは、このファイルは app.config ファイルの AllowConfigurationOverride でオーバーライドできます。このオーバーライドは、次の場合に使用します。
    - 複数の開発者が単体テストを実行し、それぞれがローカルにデータベースを持っている
    - 単体テストを実行し、異なるテスト データベースを持つビルド コンピューターが使用されている

- この機能を使用する場合は、次の手順を実行します。
    - 単体テスト プロジェクトの app.config ファイルで「<DatabaseUnitTesting AllowConfigurationOverride="true" >」と入力します。
    - 単体テスト プロジェクトで新しい .config ファイルを作成します。ファイル名は次のいずれかにする必要があります。
        - <コンピューター名>.dbunittest.config
        - <ユーザー名>.dbunittest.config
    - <DatabaseUnitTesting> を含めて、任意の接続文字列を定義します。
    - local.testsettings を開いて [配置] タブをクリックします。
    - 配置項目をクリックして有効にします。
    - 構成ファイルに追加します。

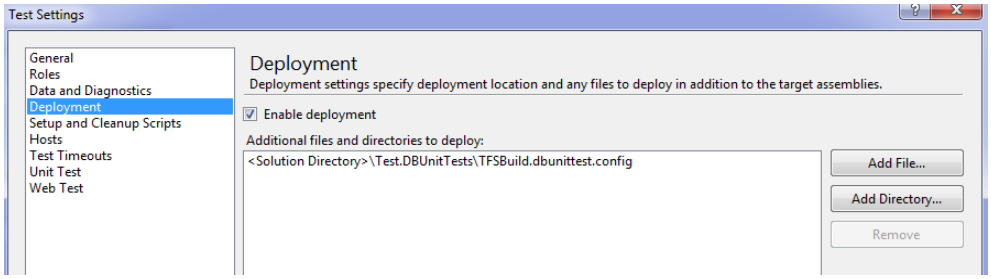


図 24: 単体テストでの配置の有効化

単体テストが実行されると、構成ファイルはテスト フォルダーに配置され、次の優先順位に基づいてテストで利用されます。

1. AllowConfigurationOverride が app.config で true に設定されているかどうかを確認します。
    a. 設定されていない場合: app.config が使用されます。
    b. 設定されている場合: オーバーライド セクションを探します。
2. 単体テストが実行されると、オーバーライドの状態を確認します。
    a. オーバーライドがコンピューターに存在している (<コンピューター名>.dbunittest.config)

b. オーバーライドがユーザーに存在している (<ユーザー名>.dbunittest.config)

3. どちらの .config ファイルも存在せず、AllowConfigurationOverride が true に設定されている場合、テストは失敗します。

配置のオプションを構成する方法

データベースがビルドの構成要素として配置される方法を変更するには、次の手順を実行します。

1. プロジェクトのプロパティで [配置] タブを選択します。

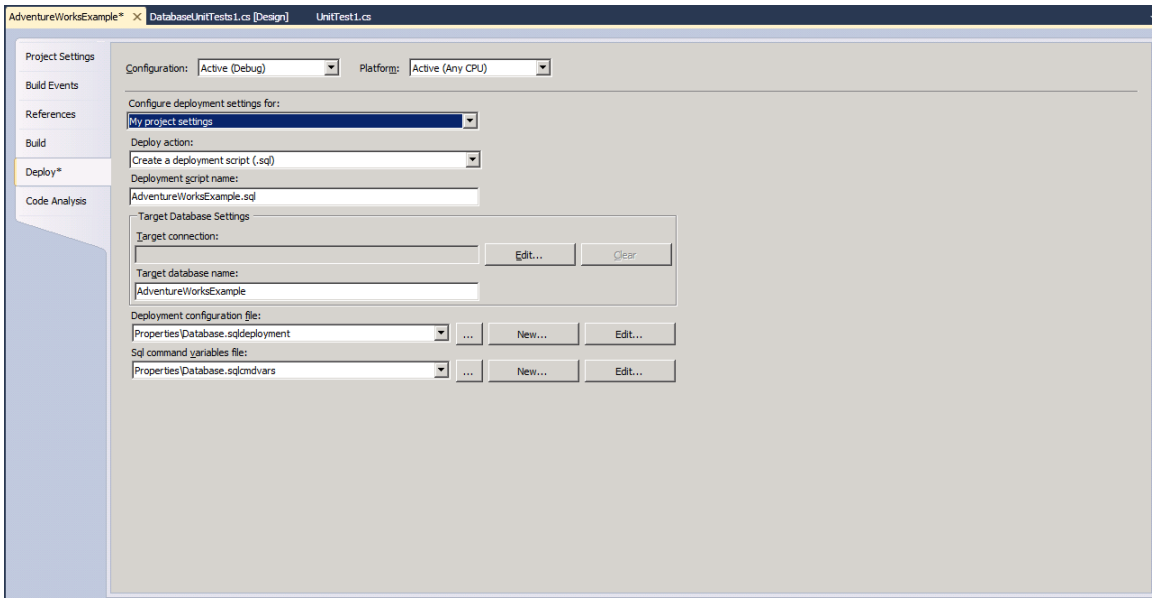
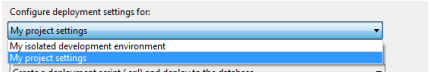


図 25: ビルドの配置オプションの変更

2. 上図の画面で、[マイ プロジェクトの設定] を選択します。ドロップダウン リストには 2 つのオプションが表示されます (下図参照)。[マイ分離開発環境] を選択すると、ローカルの "サンドボックス" 開発環境の設定を構成できます。このオプションを選択すると、自分の開発環境固有の設定を指定できます。[マイ プロジェクトの設定] は、チームに共通の設定 (ビルド コンピューターで使用されるデータベース構成設定など) を指定する場合に使用します。



3. これで、データベース プロジェクトの配置設定を構成できるようになりました。
4. [配置構成ファイル] の横にある [編集] をクリックします。

5. 配置構成ファイルのオプションが表示されます。
    a. 継続的インテグレーション ビルドの場合は、[データベースを常に再作成する] チェック ボックスがオフになっていることを確認します。CI ビルドでは、ビルドの速度を確保することが重要です。速度が速ければ、変更のフィードバックを開発者に適時提供することができます。また、データベースを削除および再作成する必要性をなくすことで、コストが高い処理もビルドから取り除かれます。
    b. 夜間ビルドの目的は、運用環境への配置をシミュレートすることです。プロジェクトには次の 2 種類があります。
        i. **グリーン フィールドのプロジェクト (データベースが存在しないプロジェクト)**: [データベースを常に再作成する] チェック ボックスは必ずオンにします。このチェック ボックスをオンにすると、ビルドでは、毎回データベースを再作成して、運用環境への配置がシミュレートされます。
        ii. **ブラウン フィールドのプロジェクト (運用データベースが現在存在しているプロジェクト)**: [データベースを常に再作成する] チェック ボックスは必ずオフにします。また、運用データベースのコピーが、ターゲット データベースとして使用され、データベースの配置が行われる前に既知の状態から復元される必要があります。この操作により、データベースが運用環境に昇格されるときに発生する可能性がある問題を特定できます。

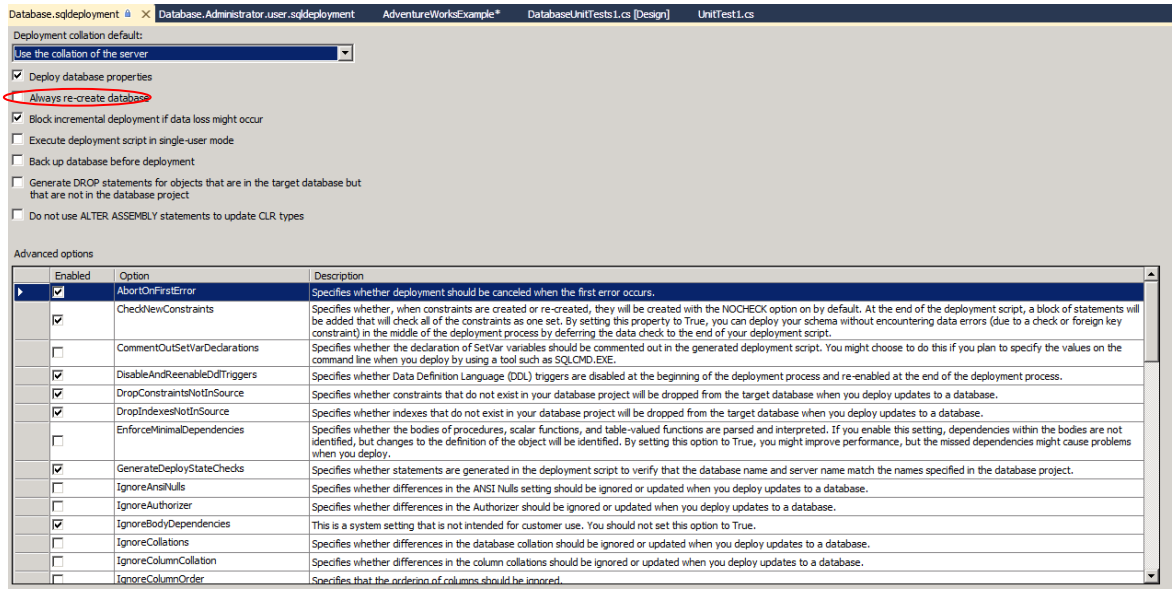
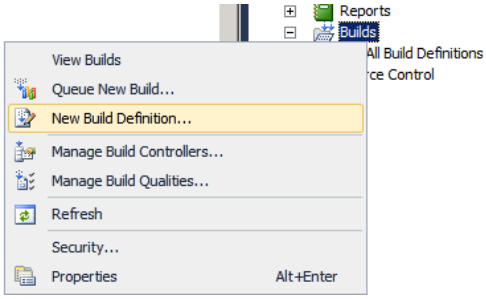


図 26: 配置オプションの指定

チーム ビルドを作成する方法

次に手順の概要を示します。

1. チーム エクスプローラーの [ビルド] フォルダーから、新しいビルド定義を作成します。



2. ビルド定義を設定してビルド プロセスを定義します。

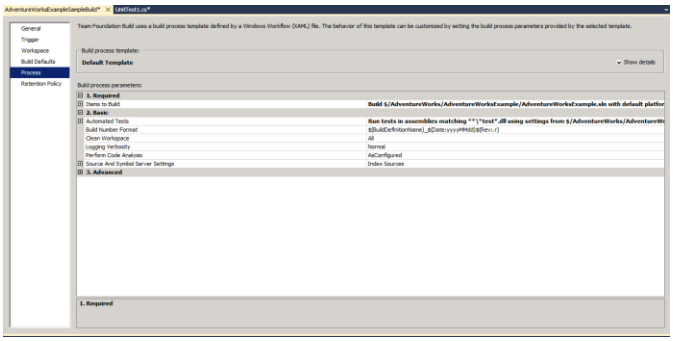


図 27: ビルド定義の設定

3. [プロセス] タブで、ビルド時に実行するテストを選択します。

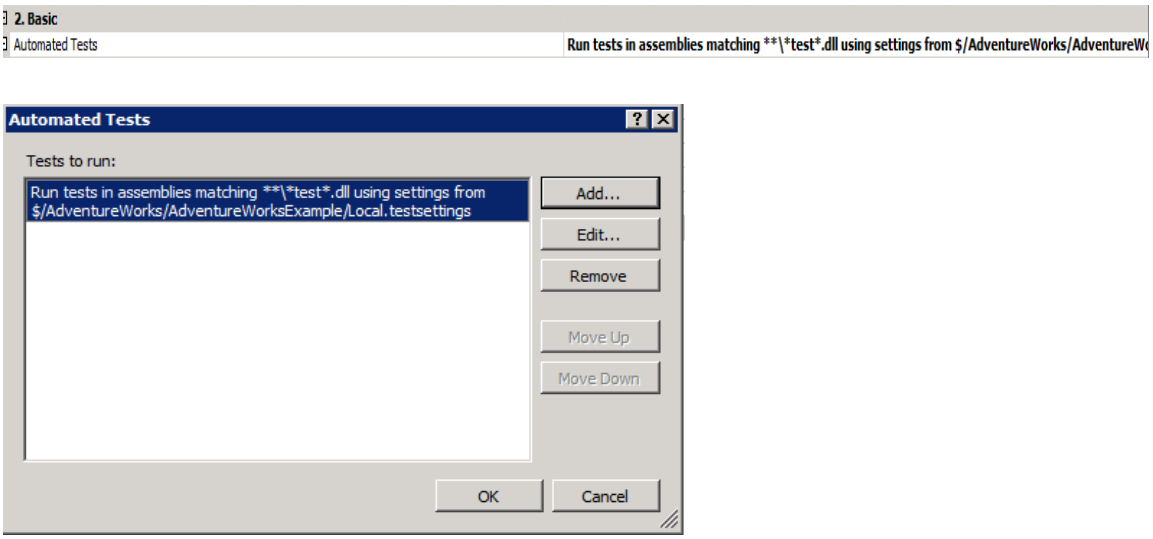


図 28: 実行するビルドのテストの指定

4. テストを行う新しいビルドを保存してキューに入れます。

データベースへの配置

生成されたスキーマ ファイル (.dbschema) は、Visual Studio データベースが既存のデータベースを比較および配置するのに使用できます。これは、IDE とコマンド ラインから実行することができます。自動化されたシナリオを実装する方法を解説する前に、"比較" 操作が行われるとき、バックグラウンドで何が起こっているかを見てみましょう。

比較操作のしくみは、次のとおりです。

1) (.dbschema で表される) データベース プロジェクトとターゲット データベース両方のモデルが作成されます。
2) この 2 つのモデルが比較され、差分が計算されます。この差分には、プロジェクトと整合性がある状態にするためにターゲット データベースに反映する必要がある変更点がすべて含まれています。
3) 差分は、配置エンジンによってターゲット データベースに適用されます。差分が適用されない場合、SQL の差分スクリプトが生成されます。

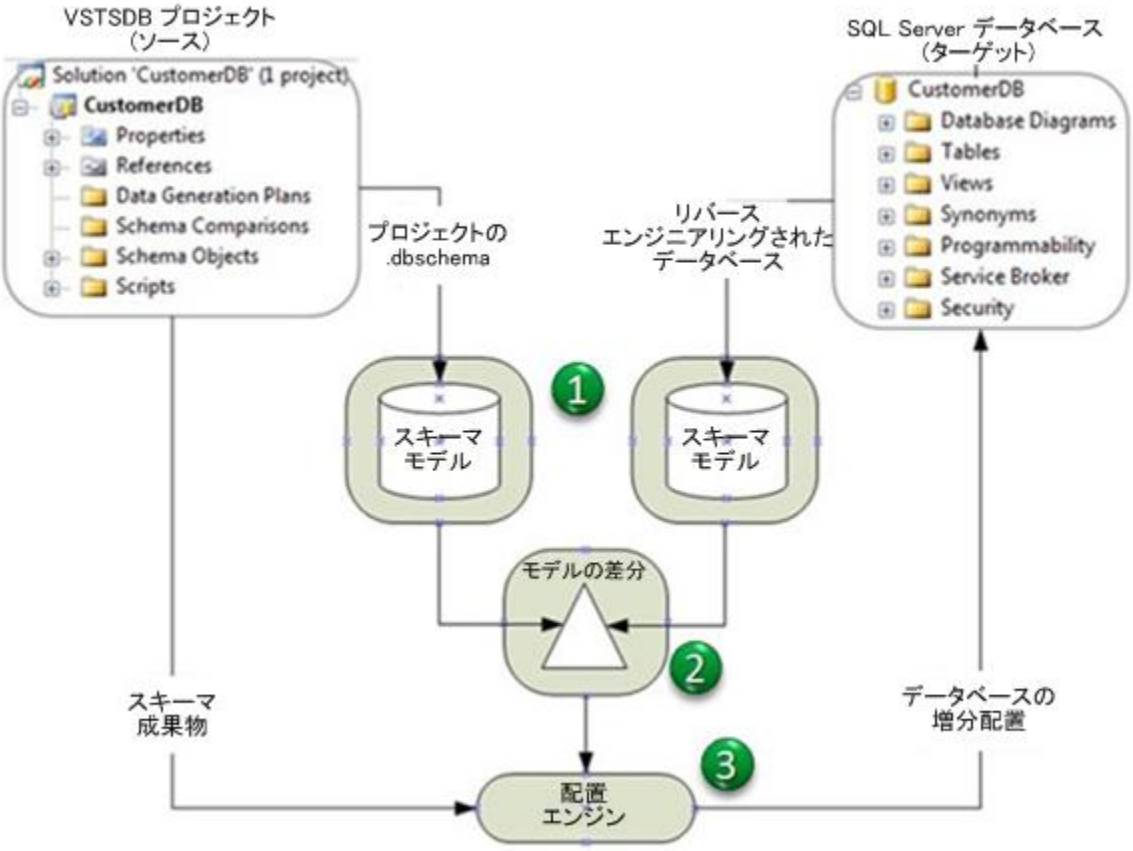


図 29: データベースの配置フロー

比較と配置を自動化するため、データベース プロジェクトには、MSBuild だけでなく、VSDBCMD というコマンド ライン ユーティリティのターゲットがあります (VSDBCMD については、次のセクションで説明します)。

VSDBCMD に .dbschema ファイルとデータベースを渡すと、データベースを .dbschema と同期するために必要なすべての SQL ステートメントを含む、増分の "差分" SQL スクリプトを生成できます。

前提条件

VSDBCMD を使用するには、コマンド ライン ユーティリティを実行するコンピューターに、いくつかの前提条件となるものがインストールされている必要があります。.NET Framework 4 など、インストールが必要なものもあれば、単にコマンド ライン ユーティリティと共にコピーするだけでよいものもあります。前提条件の詳細については、MSDN の記事「方法: VSDBCMD.EXE を使用してコマンド プロンプトからデータベースの配置を準備する[27]」を参照してください。

次に、.dbschema ファイルをデータベースと比較して増分スクリプトを生成する例を示します。

```
VSDBCMD.exe /a:Deploy /cs:"Data Source=MyServerName;Integrated
Security=True;Pooling=False" /dsp:Sql
/model:c:¥AdventureWorks¥AdventureWorks.dbschema /p:TargetDatabase=
AdventureWorks /p:SqlCommandVariablesFile=
c:¥AdventureWorks¥Properties¥Database.sqlcmdvars /manifest:
c:¥AdventureWorks¥AdventureWorks.deploymanifest /script:
c:¥ AdventureWorks¥AdventureWorks.sql
```

VSDBCMD から直接データベースに変更を適用することは可能ですが、増分スクリプトを最初に確認して、後から実行する必要がある場合もあります。つまり、VSDBCMD では、増分配置のためのスクリプトだけを作成すること可能です。差分スクリプトを作成して後から実行するというこの 2 ステップの手法は、多くのシナリオにおいてメリットがあります (詳細については、後述します)。

[27] http://msdn.microsoft.com/ja-jp/library/dd193258.aspx

次に示す、結果として生成される SQL スクリプトは、対応するサーバーで SQLCMD コマンドを実行しているターゲット データベースで簡単に実行できます。

```
sqlcmd –ic:\AdventureWorks\AdventureWorks.sql
```

次に示す、結果として生成される SQL スクリプトは、対応するサーバーで SQLCMD コマンドを実行しているターゲット データベースで簡単に実行できます。`/dd:+` を追加するだけで、スクリプトを直接配置できます。

VSDBCMD の詳細なコマンド ライン リファレンスについては、「VSDBCMD.EXE コマンド ライン リファレンス (配置およびスキーマのインポート)[28]」を参照してください。

これで、基本的なエンド ツー エンドのビルドと配置の自動化を実装するために必要なものが揃いました。

次のセクションでは、これらの概念に基づいて、複数環境への配置をサポートする方法を説明します。

複数環境設定でのビルド プロセスと配置プロセス

このセクションでは、次のことについて説明します。

- 複数環境または複数サーバーとビルドの昇格シナリオ
- 1 回のビルドで複数回配置する方法
- 複数のターゲットに配置するための差分スクリプトを生成する方法

開発組織の構成によっては、さまざまな検証目的でビルドをステージングするのに使用するテスト環境が 1 つ以上必要になることがあります。

マイクロソフトの IT チームが、複数のデータベース サーバーがある 6 ～ 7 個のテスト環境を使用することは珍しくありません。通常、ビルドは、特定の品質基準を満たすと、次の環境に昇格されます。この特質により、状態や成熟度が異なる複数のデータベースが存在することになります。たとえば、環境 A に配置されたビルドの品質が不十分な場合、ビルドは環境 B には昇格されず、環境 A と環境 B にあるすべてのデータベースが同期されていない状態になります。当然のことながら、このシナリオは、環境の数によって増加します。次の図表は、マイクロソフトの小規模な IT プロジェクトにおける典型的なビルド昇格のプロセスを表したものです。

[28] http://msdn.microsoft.com/ja-jp/library/dd193283.aspx

一般的な課題は、すべての環境が同じに見えるわけではないことを考慮せずに、どの環境にも簡単に配置できるビルドを作成する方法です。

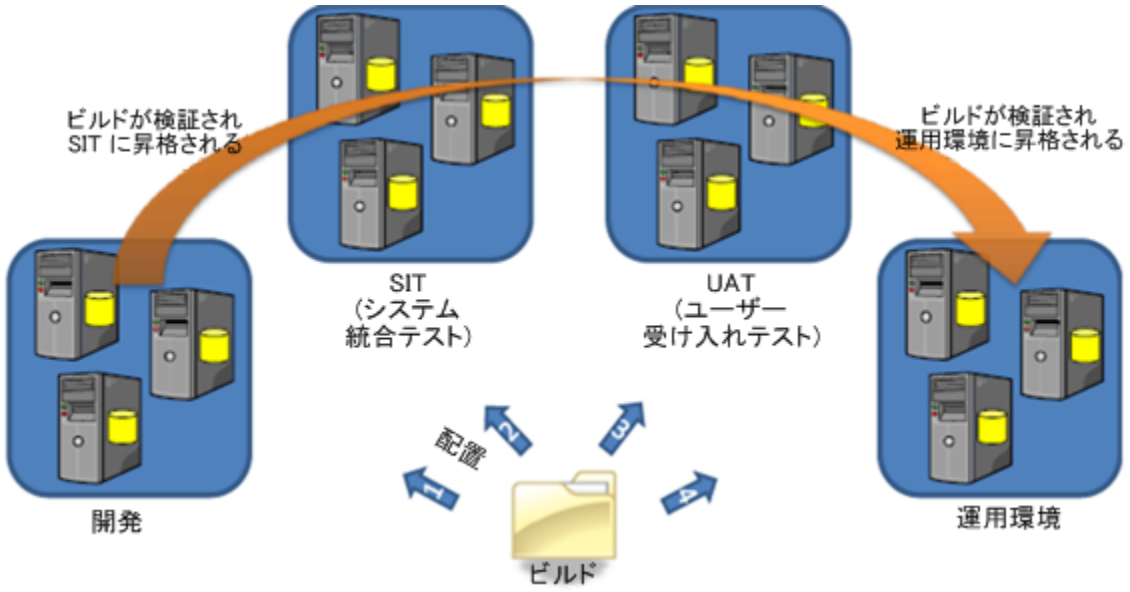


図 30: あらゆる環境に配置できるビルドの作成

マイクロソフトの IT チームのほとんどは、ビルド プロセスと配置プロセスを明確に区別しています。通常、ビルド プロセスでは、すべての Visual Studio プロジェクトとソリューションをコンパイルして、"格納" と呼ばれるものを生成します ("ビルド" とも呼ばれます)。格納には、さまざまな形があります。単なるファイルのコレクションの場合もあれば、さまざまなテクノロジによってパッケージ化されている場合もあります。

つまり、配置プロセスとは、格納のコンテンツを使用して、ターゲット環境に含まれているあらゆるサーバーにすべての必要な変更を追加することです。

図 31: サーバーのチーム ビルド

1 つ前のセクションで説明したように、たとえば、チーム ビルドを使用してビルドと配置のプロセス全体を一括で実行することは可能ですが、複数のテスト環境が使用されていて、特定の環境にインストールされる前にビルドになんらかの処理を施す必要がある場合は、これらの処理を一括で実行することはできません。この状況では、ビルドと配置は 2 つの個別のプロセスになります。

Microsoft Team Foundation Server のチーム ビルドは、格納場所の既定の出力に .dbschema ファイル、配置マニフェスト、配置オプション、配置前/配置後スクリプトが含まれているビルド プロセスを実行するための、一般的なビルド自動化システムです。

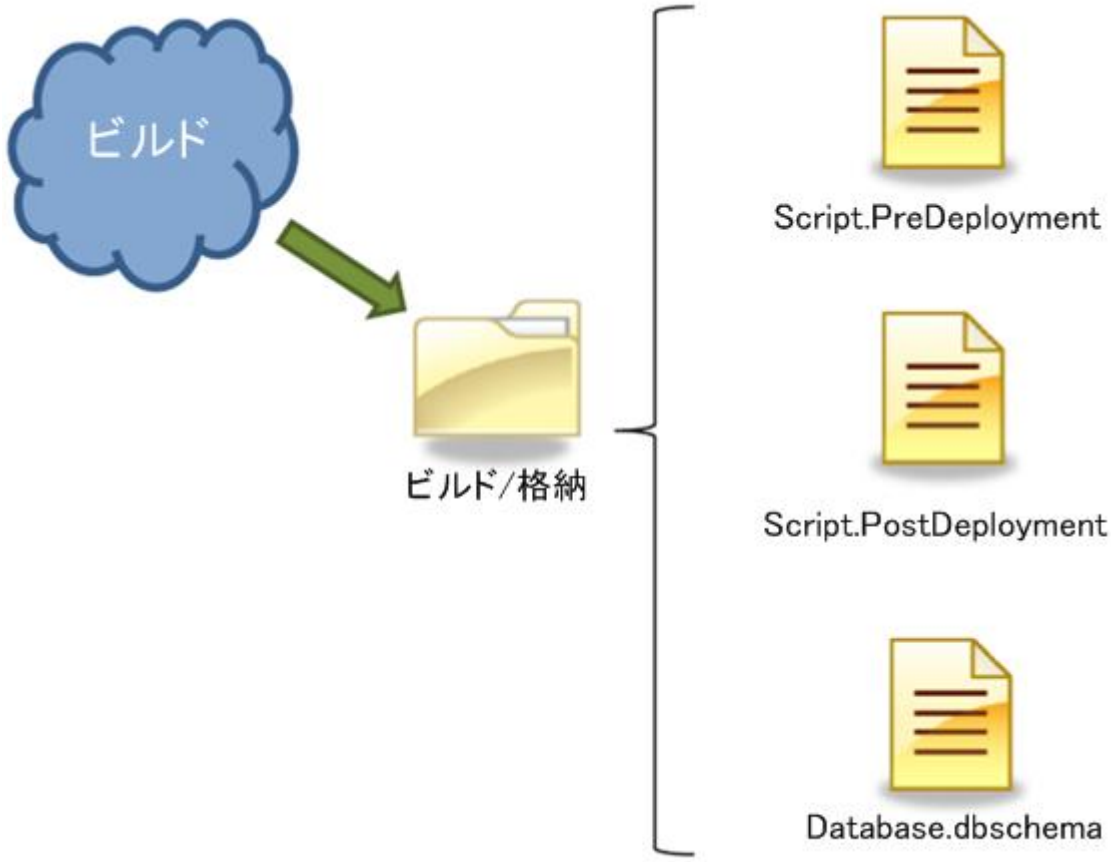


図 32: VSDBCMD と格納場所を使用したビルドの自動化

MSBuild、.BAT、または他のスクリプト言語で簡単な配置スクリプトを作成するために必要なファイルは、これだけです。配置スクリプトの基本的なアルゴリズムは、次のとおりです。

1) 格納場所とターゲット データベースを入力として使用し、VSDBCMD.exe を実行します。結果として生成される差分スクリプトを既知の場所にリダイレクトします。
2) SQLCMD を使用して差分スクリプトを実行します (または、VSDBCMD `/dd:+` オプションを使用して、スクリプトを自動的に配置します)。

お使いの環境にサーバーが多数ある場合、配置スクリプトのコンテンツをコピーして、配置スクリプトをローカルで実行することを検討してください。このようにすると、並列実行が可能になり、スキーマの比較時間を短縮できます。このためには、格納をコピーしてサーバーでリモート実行を行います。手動でも実行できますが、Windows PowerShell などのユーティリティによる自動化された方法を使用することをお勧めします。

前述のビルドの配置シーケンスでは、差分スクリプトが生成され、すぐに実行されます。この処理が必要かどうかは、開発組織の構成によって異なります。

データベースの管理者が一部の環境を所有していて、実行されるすべての SQL ステートメントが厳密に検証されない限りはデータベース上で何も実行させない場合があります。このシナリオをサポートするには、差分スクリプトを事前に生成します。つまり、基本的に、スキーマ比較操作を配置プロセスからビルド プロセスに移動します。

この考えは、チーム ビルドに次のことを実行させるというものです。

1. Visual Studio データベース プロジェクトをコンパイルする
2. 次のものに対してスキーマ比較を実行する
    a. 環境のデータベース (差分スクリプトを生成する)

注意事項

この方法の欠点は、ビルド プロセス (ビルド コンピューターなど) が、環境に依存する昇格可能なビルドを作成するために、すべてのデータベースまたはすべての環境に接続する必要があることです。これにより、ビルドのパフォーマンスに影響が及び、ビルド時に環境のデータベースでトラフィックが発生する可能性があります。そのようなリスクをなくすには、次の方法を使用します。

    b. ターゲット モデル ファイル (差分スクリプトを生成する)

留意事項

Visual Studio 2008 GDR 以降、データベース プロジェクトには、.DBSchema ファイルにデータベースをインポートする新機能が用意されています。これは、スキーマ ファイルに環境のデータベースをインポートするのに使用できます。チーム ビルドと VSDBCMD のどちらでも、これらのスキーマ ファイルをソース コントロールからビルドされたものと後から比較して、増分スクリプトを事前に生成できます。この方法は、ビルド サーバーとデータベースが異なるサブネットワークかドメインに存在していて、互いが見えない場合に特に便利です。

3. 配置スクリプトで直接実行できる差分スクリプトを含む格納を作成します。

この手法のメリットは、データベース管理者と他の関係者が、格納にあるすべての差分スクリプトをすぐに確認できることです。

最初に配置の成果物を確認する必要があるのは、データベース管理者だけではありません。開発中およびテスト中に想定されている運用データベースのスキーマが、配置時のスキーマと一致していることを確認するのは、開発者とテスターの利益にもつながります。したがって、運用データベースを実際に変更する前に、これらのモデルのずれを見つけることは非常に重要です。テストと配置に関するセクションで配置の検証について説明している箇所では、その詳細の前にずれを確認する方法について解説しています。

自動化に関する他のトピック

VSDBCMD と WiX によるデータベースの配置の自動化

VSDBCMD はデータベース スキーマを適切に配置するので、任意のスクリプト作成ツールを使用して、比較的簡単に配置スクリプトを作成できます。ですが、より完全なインストールを実行する場合は、Windows インストーラーを使用することをお勧めします。Windows インストーラーを使用すると、ソース コードやデータベース スキーマなどのあらゆる開発の成果物をまとめてパッケージ化したり、配置のリリースをバージョン管理したり、バージョン管理しているリリースを WMI/MOM で検出できるようにしたり、インストール前にソフトウェアの前提条件を確認したり、データベースに関連しているエラー処理と設定ワークフローのデザイン機能を提供する環境構成設定を処理したりすることができます。

これ以降では、かなり高度なシナリオについて見て行きます。ですが、その前に、Windows インストーラー XML (WiX) が、Windows インストーラーの *. msi ファイルを作成するための XML 言語であることに言及しておきます。このガイドは WiX のチュートリアルではないので、データベース プロジェクトと WiX 間の統合ポイントに焦点を当てます。WiX と Windows インストーラー SDK に詳しくない場合は、ここで説明していることを実践する前に、オンラインのチュートリアルを参照することをお勧めします。たとえば、私は、WiX と Windows インストーラーの新しい分野について理解したいときには、Alex Shevchuk が提供している便利なチュートリアル[29] を参照しています。

この例では、非常に軽量なデータベース プロジェクトをいくつか使用します。経験上、プロジェクト スキーマのサイズと複雑さは、インストールの複雑さとは関係ありません。重要なのはソリューションの構造と、さまざまな種類のプロジェクト間の連携です。WiX に関する共通の理解と物理的な性質に集中していただくために、すべての注意

[29] http://blogs.technet.com/alexshev/pages/from-msi-to-wix.aspx (英語)

事項については、なるべく複雑にならないように配慮しました。機能について説明するために使用するプロジェクトは、次のとおりです。

- **Customers**: 少数のテーブル ビューとストアド プロシージャがある、シンプルな顧客データベースです。
- **Sales**: Customers データベースを参照するシンプルな受注データベースです。
- **CustomerServer**: リンク サーバーと SQL エージェント ジョブを管理するサーバー プロジェクトです。
- **CustomerCLR**: Customers データベースにいくつかのカスタム型を定義する SQLCLR プロジェクトです。
- Sales と Customers はどちらもシステム オブジェクトを参照し、master.dbschema への参照があります。

これらをまとめてインストールにパッケージ化するときに失敗の原因となる統合ポイントの概要は次のとおりです。

- master.dbschema はインストールに含めないと、VSDBCMD の実行でエラーが発生するおそれがあります。
- 同じ理由から、Customers.dbschema も Sales データベースのインストールに含める必要があります。
- CustomersServer データベースは、Customers データベースより先にインストールする必要があります。Customers データベースは、Sales データベースより先にインストールする必要があります。サーバー プロジェクトに属するサーバー規模の設定である CLR サポートを有効にするのは、データベース プロジェクトではなく、CustomerServer です。
- CustomerServer プロジェクトを配置する前に、SQLAgent サービスの実行を停止して再起動します。存在しないデータベースを参照する SQL ジョブを作成しないようにします (このような SQL ジョブを作成するとエラーになります)。
- Cusotmers データベースと Sales データベースは、データとログのパスを公開して、インストール時に設定できるようにする必要があります。

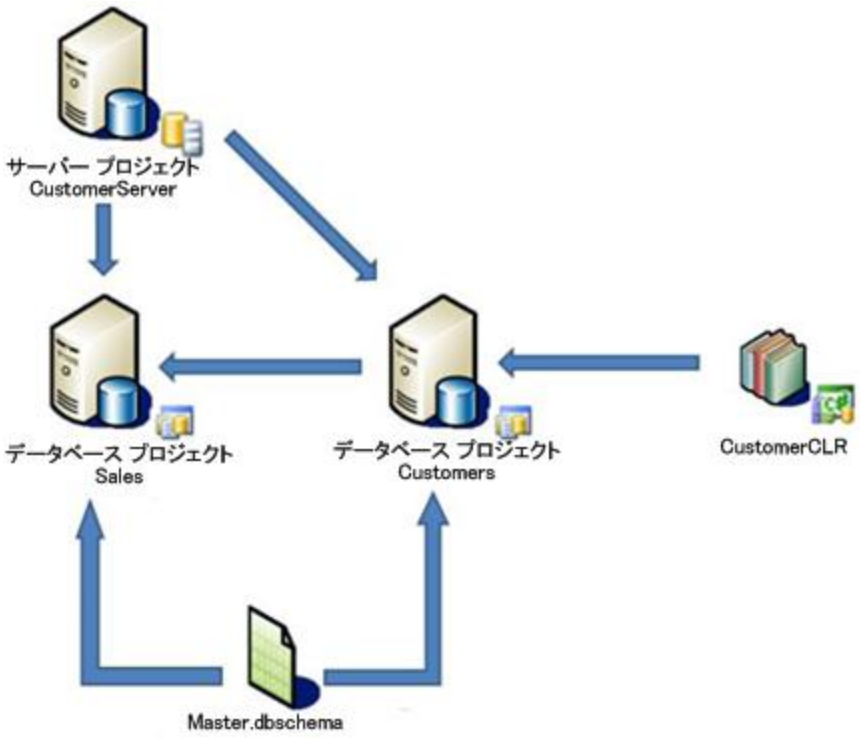


図 33: 配置の依存関係

さらに、VSDBCMD の再頒布可能ファイルをパッケージ化して、すべてのデータベースのインストール前にインストールする必要があります。

WiX インストーラー プロジェクトの実行可能な機能とセマンティクスの概要、およびこのシナリオに基づいて独自のインストーラー プロジェクトを作成する方法の詳細について説明しました。

配置構成と配置前/配置後スクリプト

データベース プロジェクトには、配置構成を制御するファイル一式が用意されています。これらのファイルは、データベース プロジェクトの [Properties] フォルダーに格納されており、配置の動作を決める一連の変数を含んでいます。

ファイルの名前と目的は、次のとおりです。

ファイル名	説明
Database.sqlcmdvars	プロジェクトを配置するときに使用する、SQLCMD 変数の名前と値を含んでいます。1 つ以上の .sqlcmdvars ファイルを定義して、各ソリューション構成を 1 つの .sqlcmdvars ファイルと関連付けます。詳細については、「方法: データベース プロジェクトの変数を定義する」を参照してください。
Database.sqldeployment	データベース名やターゲットの接続文字列など、配置固有の設定を含んでいます。1 つ以上の .sqldeployment ファイルを定義して、各ソリューション構成を 1 つの .sqldeployment ファイルに関連付けます。これらのプロパティを構成する方法の詳細については、「方法: データベースおよびサーバー プロジェクトの配置設定を構成する」と「方法: 配置の詳細のプロパティを構成する」を参照してください。

オプションの詳細については、MSDN の記事「データベース プロジェクトおよびサーバー プロジェクト内のプロパティ ファイル[30]」を参照してください。

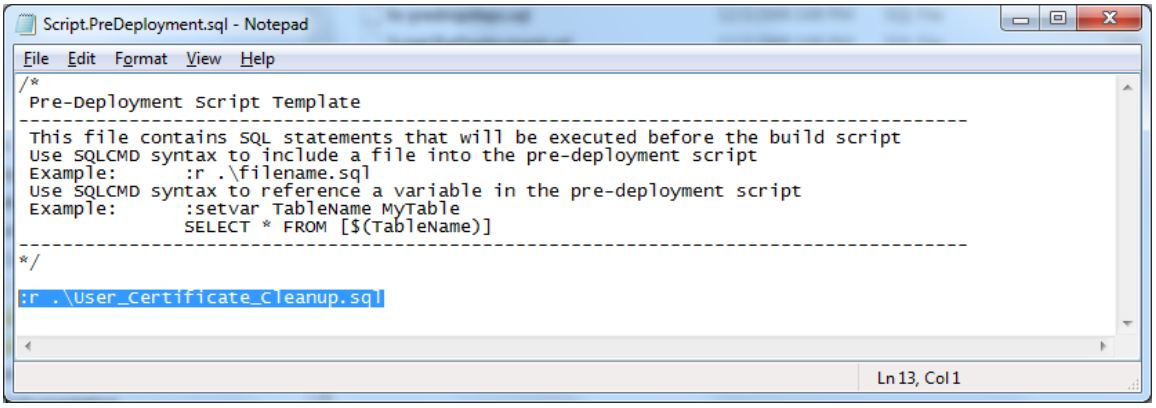
[30] http://msdn.microsoft.com/ja-jp/library/dd193289.aspx

配置前/配置後スクリプトと再実行

データベース プロジェクトを採用している組織のほとんどは、データベースのシードや特定のシナリオへの対処など、さまざまな目的に使用する非スキーマ スクリプトを数多く保有しています。

データベースが新しいデータベース プロジェクトにインポートされたら、これらのスクリプトは、配置中に実行する必要があるタイミングによって、配置前スクリプトまたは配置後スクリプトとして手動で配置できます。データベース プロジェクトでは、[Scripts] フォルダーにある Script.PreDeployment.sql ファイルと Script.PostDeployment.sql ファイルに、これらのスクリプトへの参照を追加して、スクリプトを追跡します。

Script.PreDeployment.sql のサンプルを次に示します。参照されているファイルを強調表示しています。



プロジェクトをコンパイルすると、配置前ファイルと配置後ファイルで参照されるすべてのスクリプトのコンテンツが、一連の新しい Script.PreDeployment.sql ファイルと Script.PostDeployment.sql ファイルに書き込まれます。これらのファイルは、.DBSchema ファイルと共にプロジェクト出力に格納されます。

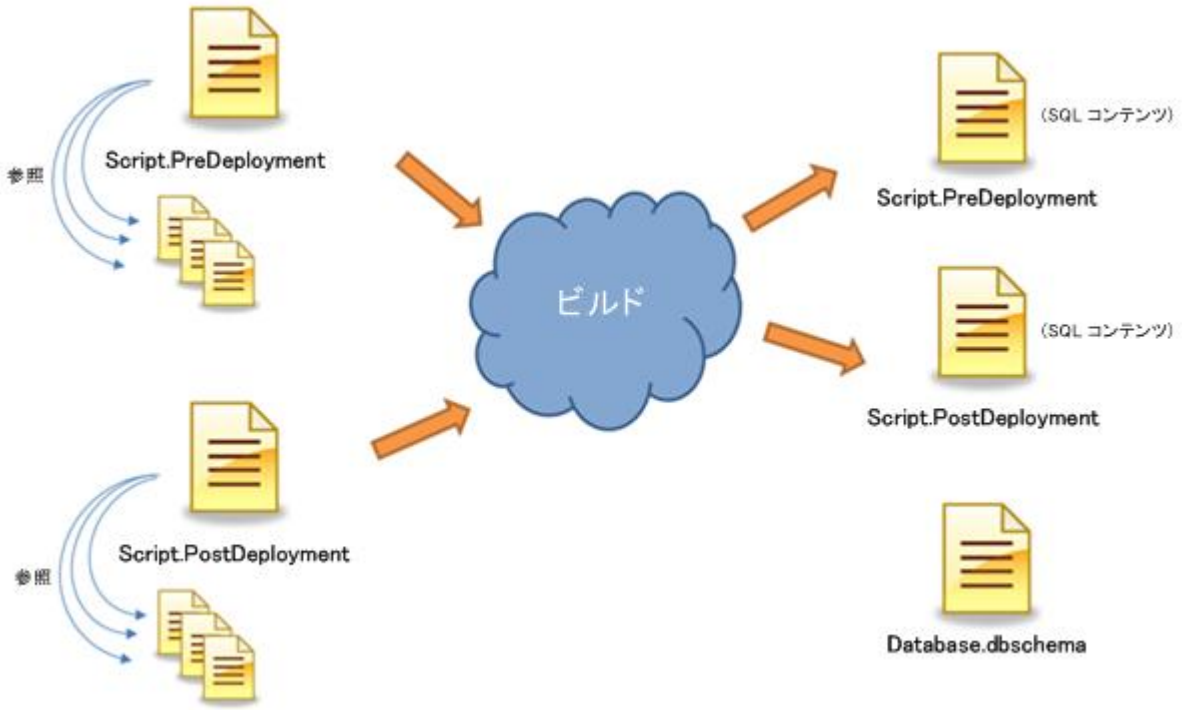


図 34: ビルド プロセスにおける配置スクリプトのシーケンス

配置時には、ビルドによって生成される配置前スクリプトと配置後スクリプトのコンテンツが、結果として生成される差分スクリプト内の比較によって生成されたコードの前後に追加されます。重要なのは、コンテンツは配置が行われている間は変更されず、そのまま差分スクリプトの最初と最後に追加されるということです。

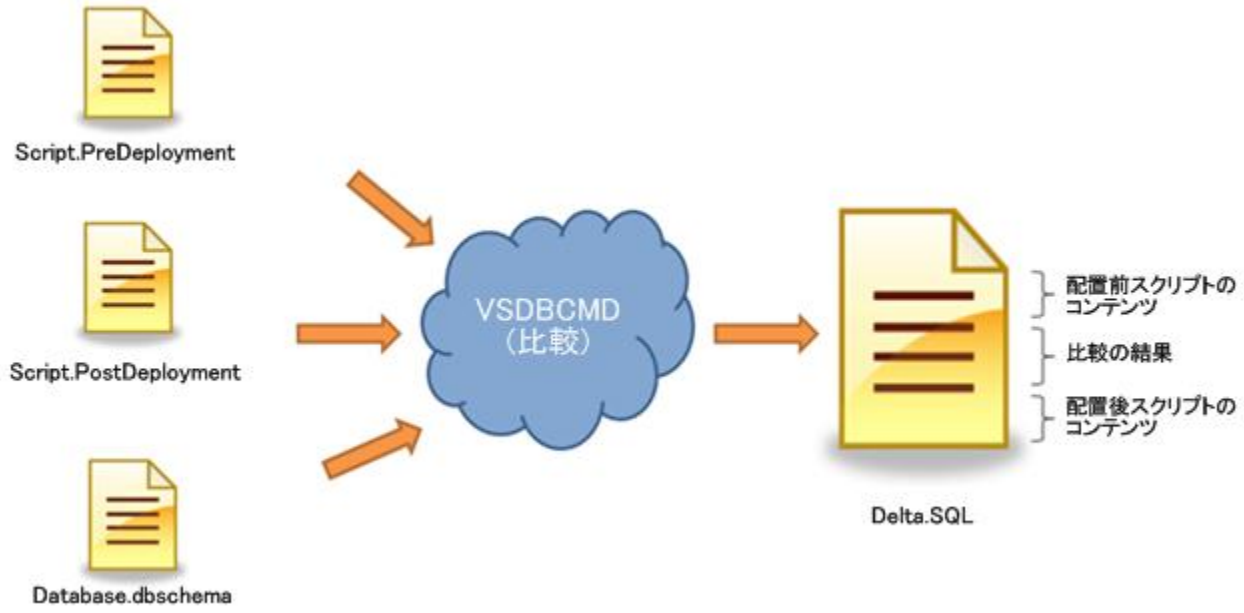


図 35: VSDBCMD による比較の出力

特定の 1 つの環境で同じデータベースに複数の配置を実行する場合、配置前スクリプトと配置後スクリプトは再実行されます。望ましくない結果を回避するには、すべてのスクリプトを、再実行可能にしたり、何度行っても結果が同じになるようにする必要があります。これはつまり、すべてのスクリプトを、再実行することによってデータベースが一貫性のない状態にならないようにデザインする必要があるということです。たとえば、スクリプトでテーブルにデータを挿入している場合、2 回目に挿入しようとする前には、データが存在しているかどうかを確認する必要があります。

再実行可能なスクリプトの例を次に示します。

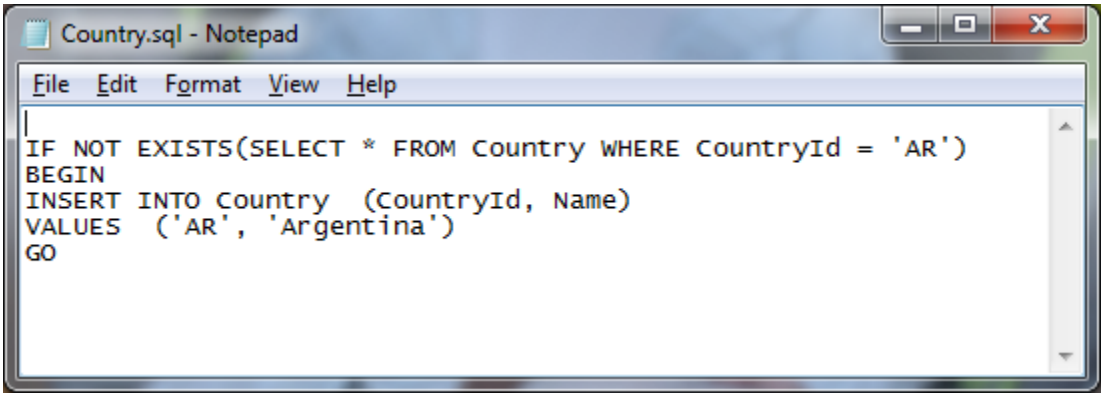


図 36: 再実行可能なテーブル スクリプトの例

推奨事項

カスタム スクリプトは、最初から、何度実行しても結果が同じになるようにして、アップグレードするデータベースがどんな状態でも、スクリプトによる影響が変化せず、最終結果が同じになるようにしてください。

チームは、一度で再実行可能にするためには多大な労力を必要とする、従来の再実行できないスクリプトを大量に保有している場合があります。また、従来の SQL スクリプトをすべて再実行可能にするために、いくつかの Visual Studio のリリースが必要な場合もあります。このような場合は、再実行可能なスクリプトもあればそうでないスクリプトもあるが、どれも少なくとも 1 回は実行する必要があるという状況に陥ります。

これは理想的な状況ではありませんが、コンパイルされてビルド出力に配置される、配置前スクリプトと配置後スクリプトを使用することで、再実行できないスクリプトに対処することは可能です。これは、Script.PreDeployment.sql ファイルと Script.PostDeployment.sql ファイルに参照を追加したり、これらのファイルから参照を削除したりすることで実行できます。

マイクロソフトのチームがこの状況に対処した方法に基づいている次のシナリオについて考えてみましょう。

チームは、既存のデータベースをホストするために約 30 個のデータベース プロジェクトを作成しました。データベースは、運用環境からインポートされ、以前はソースコード管理下にあった、すべての既存のスキーマ コードは、破棄されました。ただし、配置前ファイルと配置後ファイルとしてプロジェクトに追加された何千ものデータ操作ファイルは除きます。これらの配置前ファイルと配置後ファイルのほとんどは、再実行できませんでした。

チームは、プロジェクトをビルドしたときに、すべての配置前ファイルと配置後ファイルを含めました。これは、環境に初めて配置する場合は適切な処置ですが、2 回目以降の配置では、データベースで深刻な破損が発生する原因となります。したがって、(データがない新しい環境のために) すべての配置前/配置後ファイルを含むビルドと、(2 回目以降の配置のために) 再実行可能な新しい配置前/配置後ファイルのみを含むビルドを作成できるようにするソリューションが必要でした。

この問題は、再実行に関するガイドラインに従って、新しい配置前ファイルと配置後ファイルを記述するで解決しました。さらにチームは、ソースコード管理下にあるデータベース プロジェクトを調べて、特定の日付の後にチェックイン (変更または作成) された配置前ファイルと配置後ファイルを検出して、Scripts.PreDeployment ファイルと Scripts.PostDeployment ファイルに配置前/配置後ファイルへの参照のみを追加するルーチンを開発しました。このルーチンはチーム ビルドに統合され、ビルドでは、参照日が提供されない場合、すべての配置前スクリプトと配置後スクリプトを含む出力が作成されるようになりました (このビルドは、新しい環境への最初の配置に使用されます)。日付が提供された場合には、新しい、または変更された配置前ファイルと配置後ファイルのみを含む出力が作成されました。この日付は、それ以降のすべての格納に使用されます。

参照日は、必要に応じて、チーム ビルドのパラメーターとして渡したり、ビルド ワークフローでじかに構成したりすることが可能です。

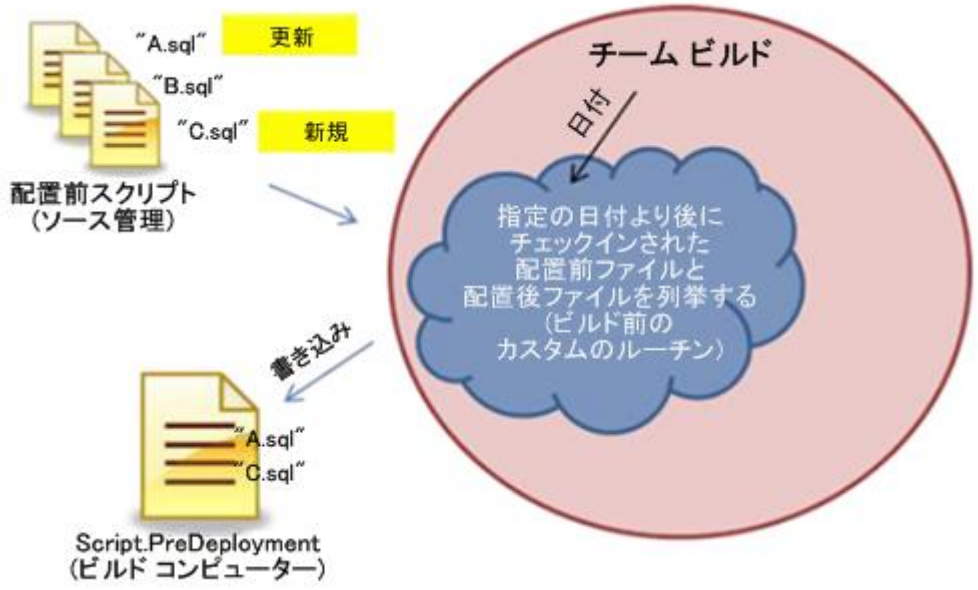


図 37: 同じ種類のソリューションを事前にフィルター処理または後からフィルター処理するスクリプト

配置前スクリプトと配置後スクリプトをフィルター処理する別の基準に基づいて、同じようなソリューションを作成することができます。

参考資料

ここで説明していないオブジェクトの詳細については、次のリンク先を参照してください。

- ビルド コントリビューターおよび配置コントリビューターを利用してデータベースのビルドおよび配置をカスタマイズする (MSDN)[31]
- チュートリアル: 配置計画を変更するためのデータベース プロジェクトの配置の拡張[32]
- Offline Schema Development (http://blogs.msdn.com/bahill/archive/2009/03/02/offline-schema-development.aspx、英語)
- Deploying your Database Project without VSTSDB installed (http://blogs.msdn.com/bahill/archive/2009/02/21/deploying-your-database-project-without-vstsdb-installed.aspx、英語)
- Managing data motion during your deployments (Part 1) (http://blogs.msdn.com/bahill/archive/2009/03/30/managing-data-motion-during-your-deployments-part-1.aspx、英語)
- Managing data motion during your deployments (Part 2) (http://blogs.msdn.com/bahill/archive/2009/07/02/managing-data-motion-during-your-deployments-part-2.aspx、英語)
- Visual Studio database Continuous Integration (http://blogs.msdn.com/bahill/archive/2009/07/31/come-visit-revisit-the-beer-house-continuous-integration.aspx、英語)
- ビルド コントリビューターおよび配置コントリビューターを利用してデータベースのビルドおよび配置をカスタマイズする (http://msdn.microsoft.com/en-us/library/ee461505(VS.100).aspx)
- データベースおよびサーバー プロジェクトの配置設定を構成するためのガイド (http://msdn.microsoft.com/en-us/library/dd193254.aspx)

- [31] http://msdn.microsoft.com/ja-jp/library/ee461505.aspx
- [32] http://msdn.microsoft.com/ja-jp/library/ee461507.aspx

データベースのテストと配置の検証

目的

- Visual Studio のデータベース ツールを使用してデータベースの単体テストを実行する方法を説明する
- データベース アプリケーションが正常に配置されたことを検証するために使用できる手法を概説する

概要

単体テストについては、長年、多数のドキュメントで解説され、実践されてきました。しかし、データベース コードの単体テストは歴史が比較的浅く、サポートしているツールもほとんどありません。以前は、開発者個人の判断で手動テストとデータ アクセス層を通じたテストのいずれかでコードをテストしていました。どちらのメカニズムにも欠点があり、非常に時間がかかる、テスト データに整合性がない、トリガーや関数などの重要なデータベース機能が十分にテストされないなどの問題があります。

Visual Studio データベース プロジェクトには、データベースのテストをサポートするツールが用意されています。Visual Studio では、開発者やテスト担当者がアプリケーションをテストする手段を簡略化するために、データベース プロジェクトと共に多数の新しいツールが導入されました。データベース プロジェクトには、ストアド プロシージャ、関数、およびデータベースのほとんどの機能を対象とした単体テスト ツールが用意されています。Visual Studio に新しく追加されたテスト ツールを使用するときには、負荷とデータ ボリュームの両方に対するパフォーマンス テストをサポートするデータ生成ツールも使用できます。

このセクションでは、次の内容について説明します。

- 単体テスト向けにプロジェクトを構成する
- テストを自動化する (テスト データの作成を含む)
- データベースの配置を検証する

このセクションでは、ツールでサポートされている機能とサポートされていない機能に重点を置きます。シナリオがサポートされていない場合は、マイクロソフト社内のさまざまなチームまたはマイクロソフトのパートナーやお客様が現場で使用している手法のガイダンスを提供します。

データベースとデータ アクセス層のテスト

これまで、データ アクセスのテストは難しい作業でした。データ アクセス層の設計と実装に使用する手法は、アーキテクチャによって大きく異なることがあります。データ アクセスには、さまざまなテクノロジを使用できますが、使用できるオプションは大きく 2 つの手法に分けられます。1 つは、NHibernate や Entity Framework などのオブジェクト リレーション マッピング テクノロジを使用して、アプリケーション内でコードを使用する手法です。もう 1 つは、ストアド プロシージャ、ビュー、関数などの形で、データベース内でコードを使用する手法です。どちらの手法でも、データベース テーブルからデータを取得して、アプリケーションで使用可能な形式に整えるコードを作成します。

データ アクセスに使用する手法に関係なく、Visual Studio に用意されているデータベース ツールはデータ アクセス層のテストに役立ちます。

失敗の迅速な判断とバグの早期発見

他のコードと同様に、データベース コードの単体テストを行うと、開発サイクルの早い段階でフィードバックを提供できます。開発の初期段階で、単体テストを使用してバグを発見すると、全体的な開発時間が短縮し、バグが開発サイクルの後半に持ち越されることを回避できます。開発およびテスト サイクル中にバグを発見して修正すると、製品のリリース後にバグを発見して修正する場合に比べてコストを桁違いに抑えられると言われています。

開発およびテスト サイクル中にバグを発見して修正すると、製品のリリース後にバグを発見して修正する場合に比べてコストが桁違いに減少するとも言われています。

また、単体テストは、変化するコードに対して定期的に実行して、既存の機能に潜んでいるバグを発見するので、一連の単体テストを実行すると、開発サイクルの後半でもバグの発生を防止するのに役立ちます。

広範なカバレッジ

データベースの機能 (ストアド プロシージャやトリガーなど) を直接対象とする単体テストでは、カバレッジが広範で、アプリケーションのデータ層を通じたテストではテストできないくらい細かく広範に対応しています。データベースの機能を直接実行すると、トリガーで呼び出されたり追加のプログラミング オブジェクトの呼び出しで呼び出されたりする、特定の動作を簡単にテストできます。この結果、データを変更するビジネス ロジックの影響を回避しながら、動作を迅速にテストしてデータ関連コードの影響を判断できます。このようなビジネス ロジックには、データベース操作の前または後に実行するビジネス ロジックなどがあります。

テストの自動化

自動化が重要な理由

自動テストを行うと、コード ベースの状態について迅速かつ早期にフィードバックを提供できます。適切なコード カバレッジを備えた自動テストを使用すると、アプリケーション コードの開発が進んでも、アプリケーションで正常な機能が確実に提供されるようになります。このようなテストが、絶えず変化するコードの回帰テストという形で、アプリケーションのライフサイクル全体にもたらす価値に注目してください。

推奨事項

基本的に、すべてのテストを完全に自動化するように努めることをお勧めします。テストを完全に自動化すると、より簡単で確実な回帰テストを実行できます。自動テストは手動テストよりも頻繁に実行しますが、最終的なコストは手動テストより低くなります。

テストの自動化に投資する時間は、テストの実行のアプリケーションから得られる投資利益率 (ROI) とのバランスを取る必要があります。テスト対象のコードが絶えず変更されている場合は (コードの変更に合わせてテストが絶えず変更されているという状況下では)、コードが安定するまで手動テストで開発を進めた方が投資効率が良いことがあります。ROI を評価する際には、コード ベースの予測寿命も考慮する必要があります。開発中のアプリケーションが一時的なソリューションとして使用されている場合、短期間のプロジェクトでは、一連の自動テストに投資しても投資を回収できないことがあります。また、特定のコードがうまく機能しなかったり処理結果が正しくなかったりする場合は、そのコードが及ぼす影響も考慮する必要があります。この場合、影響の性質に応じて、自動テストまたは手動テストを追加しなければならないことがあります。

制限

自動データ テストで効果的に対応できる範囲には制限があります。多くの場合、実行にかかる時間が制限の要因になります。アプリケーション コードを対象に作成された純粋な単体テストとは異なり、データベース テストでは、多くの場合、データベースを再配置したり、データベースのテストを実行可能で意味のあるものにするために必要なデータを生成したりしなければなりません。データベースとデータのサイズが大きくなると、データベースを作成してデータを設定するのには非常に時間がかかることがあります。

開発サイクル限定でデータ生成計画を効果的に使用するには、データベースとデータ生成計画のどちらかが変更されたときには、データベースを再作成し、データ生成計画を再実行して、開発中のコードをサポートするようにします。そのため、データベースと生成計画を再作成して配置する処理は、テスト プロジェクトの実行から除外することをお勧めします。

データベースの配置とデータ生成計画は、それぞれが変更されたタイミングで、必要に応じて手動で実行します。いずれかのテストでデータが変更される場合は、トランザクション内でテストを実行し、テストが終わったらトランザクションをロールバックします。このようにすると、データベースが整合性のある状態に保たれ、テストを実行するたびにデータベースの作成やデータ生成を再実行する必要がなくなります。

現実的で明確なデータの生成は重要ですが、次の手法を採用する場合は、このようなデータの生成について慎重になる必要があります。

- さまざまなシード値を使用して、テストを実行するたびにランダムなデータを生成する
- 同じシード値を使用して、テストを実行するたびにデータを生成する
- 組み込みのデータ バインド ジェネレーターを使用して、現実的なデータや (セキュリティ上の理由から) スクランブルがかけられたデータを取得するための参照データベースを使用する

どの手法にも、明確さ、速度、現実的なデータ分布の幅広さと詳細さ、およびユース ケースとコード カバレッジの使用方法に関してメリットとデメリットがあります。採用する手法は状況に応じて決定する必要があります。ほとんどのテスト ケースでは、参照データを一度だけ作成し、外部データ ストアから取得してテストのスキーマを設定することをお勧めします。

単体テスト向けのプロジェクトの構成

データベース単体テストで使用できるテストの種類に関する詳細については、「単体テストを使用したデータベース コードの検証[33]」を参照してください。

データベース単体テストは (単体テスト、パフォーマンス テスト、または Web テストのいずれであるかにかかわらず)、他のテストの種類と同じ種類の Visual Studio プロジェクトの種類に分類されます。これは便利な反面、テストに必要なプロジェクトの数、テストに使用するプロジェクトの数、プロジェクトでテストをグループ化する方法について混乱も引き起こすことがあります。

配置またはデータ生成計画が失敗すると、プロジェクトのすべてのテストが失敗したテストとしてマークされます。この手法は、配置またはデータ生成計画が失敗した場合はデータベースに問題があるのでテストが実行されず、両方とも成功した場合はテスト結果が完全に信頼できるという前提で採用されています。同じプロジェクトに他の種類のテストが含まれている場合、他の種類のテストが当該テストの失敗の影響を受けなくても、失敗したテストとしてフラグが設定され、正しくない結果が返されます。

テストを分類して整理する主な手法は、次の 2 つです。

- ビジネス機能
- 技術機能

[33] http://msdn.microsoft.com/ja-jp/library/dd172118.aspx

どちらの分類を選択した場合でも、開発者がデータベース テストを単体テスト戦略の一環として実行する必要がある場合は、データベース テストは簡単かつ迅速に実行できる必要があります。実行できないときは、そのテストに効果がありません。実行に時間がかかるテストは、チームのビルド サーバーでスケジュールされたビルドの一環として実行するのが適しています。

ビジネス機能によるテスト プロジェクトのグループ化

ビジネス機能でテストをグループ化すると、必要なデータ生成計画をテスト対象の特定の領域に簡単に分離できます。データ生成計画を分離すると、データ生成計画でユーザー シナリオや機能の定義に沿った機能領域ごとにテストを簡単に作成して管理し、分離できるようになります。

開発チームが機能ごとのチームで作業する場合や、チーム メンバーがアプリケーションの層ごとではなく機能領域ごとに作業することが多い場合は、ビジネス機能でプロジェクトをグループ化することも役に立ちます。たとえば、一部のメンバーがユーザー インターフェイス層で作業し、他のメンバーがデータ アクセス層の開発を担当する場合に便利です。

ビジネス機能でグループ化する手法を使用してアプリケーションを開発する場合は、チーム固有のテスト プロジェクトに同じチーム メンバーが含まれるようにすると、チーム間で発生するプロジェクトの競合をソースコード管理レベルで最小限に抑えることができます。また、特定の領域を 1 つのチームで作業すると、特定のビジネス機能を簡単にデータ生成計画の対象にできます。

技術機能によるテスト プロジェクトのグループ化

技術機能でグループ化するときには、ロード テストやパフォーマンス テストなどの側面が含まれます。多くの場合、この種類のテストでは、テスト データの要件やボトルネックを特定する処理の順序が大きく異なります。

この方法でグループ化すると、1 つのアプリケーションで同時に作業している複数の大規模なチームどうしの連携を強化するのに役立つことがあります。つまり、データ アクセス チーム、サービス チーム、およびインターフェイス チームに分かれて作業している場合、テスト プロジェクトの構造をチームの構造に合わせると、共同作業が容易になり、プロジェクトの競合を最小限に抑えられます。

中規模の開発プロジェクトでデータ生成計画を活用するには、テストをビジネス機能に分割することをお勧めします。ただし、ストレス テストとロード テストでは、データ生成計画が大きく異なるため、この推奨事項は当てはまりません。

ストアド プロシージャのテスト向けテスト プロジェクトの迅速な作成

スキーマ ビューを使用すると、ストアド プロシージャの単体テストを迅速に作成できます。そのためには、まず [表示] メニューの [データベース スキーマ ビュー] をクリックします。次に、[ストアド プロシージャ] ノードが表示されるまでスキーマを展開します。ストアド プロシージャを右クリックして、[単体テストの作成] をクリックします。

表示されたウィザードの指示に従って、次の手順を実行します。

- 表示されたウィザードの指示に従って、次の手順を実行します。
- テストを作成する対象となるストアド プロシージャを選択します。
- データベースの配置、作成、およびデータ生成に関するオプションを選択します。

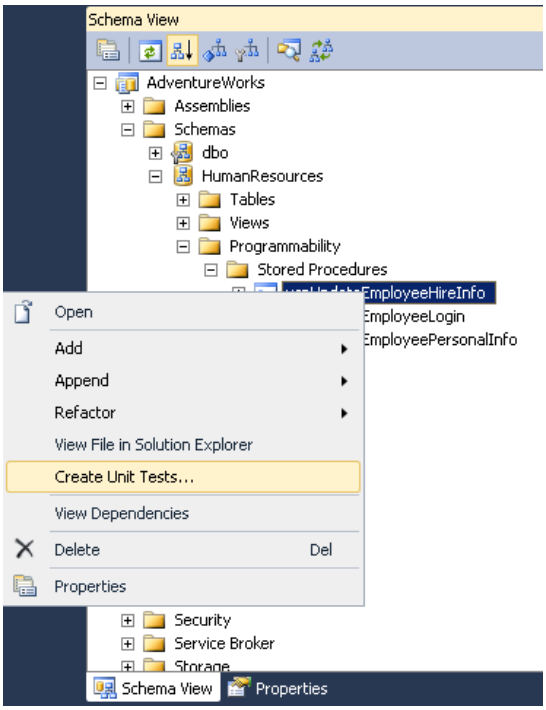


図 38: Visual Studios のコンテキスト メニューから単体テストを簡単に作成できる

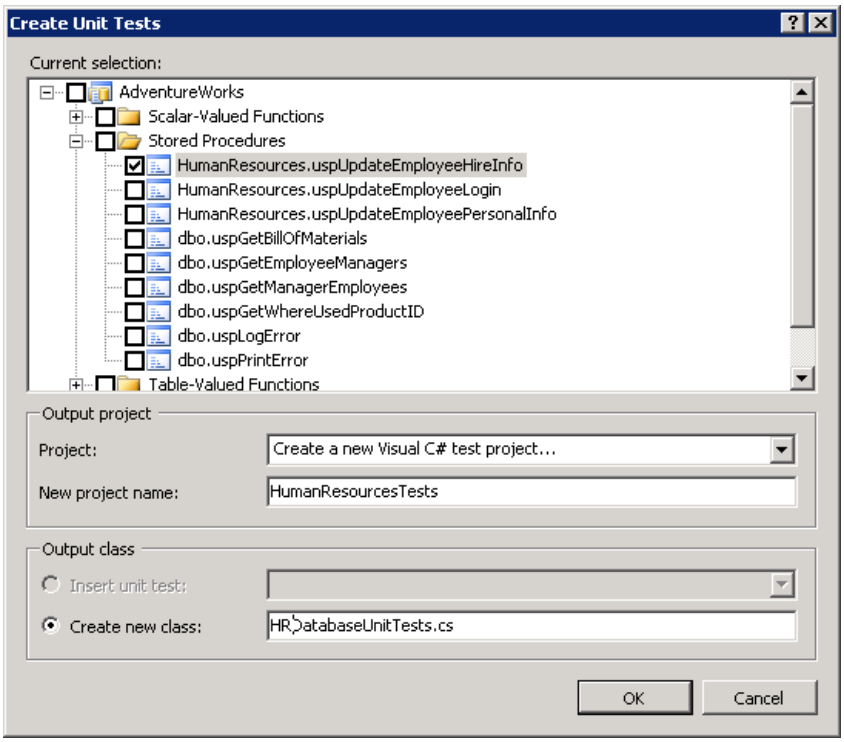


図 39: 成果物を選択すると実行用の Transact-SQL のスタブと .NET コードが生成される

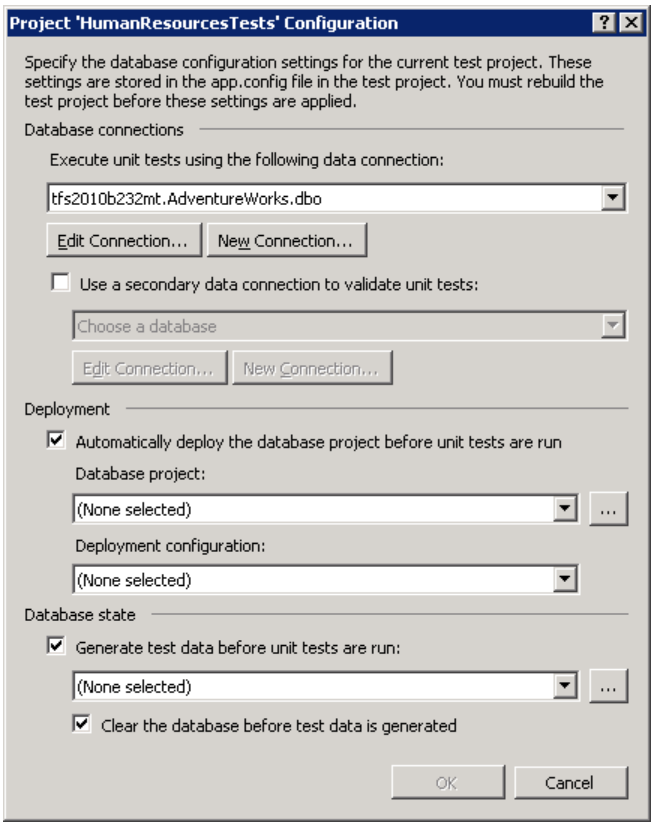


図 40: 実行、検証、および準備環境用にテストを構成できる

注

作成されるテスト プロジェクトは標準的な Visual Studio のテスト プロジェクトなので、このプロジェクトには、Visual Studio で提供される他の任意の種類のテストを追加できます。ここで重要なのは、データベース プロジェクトが独自のプロジェクトに配置されることです。このため、アセンブリ名に基づいてテストを検索する CI ビルドの一環としてデータベース プロジェクトをビルドする場合、データベース プロジェクトを分離できます。この方法は、アプリケーションの他の層を対象とするテスト アセンブリのセグメント化によってテスト ケースがより明確に分離されるときにも役に立ちます。

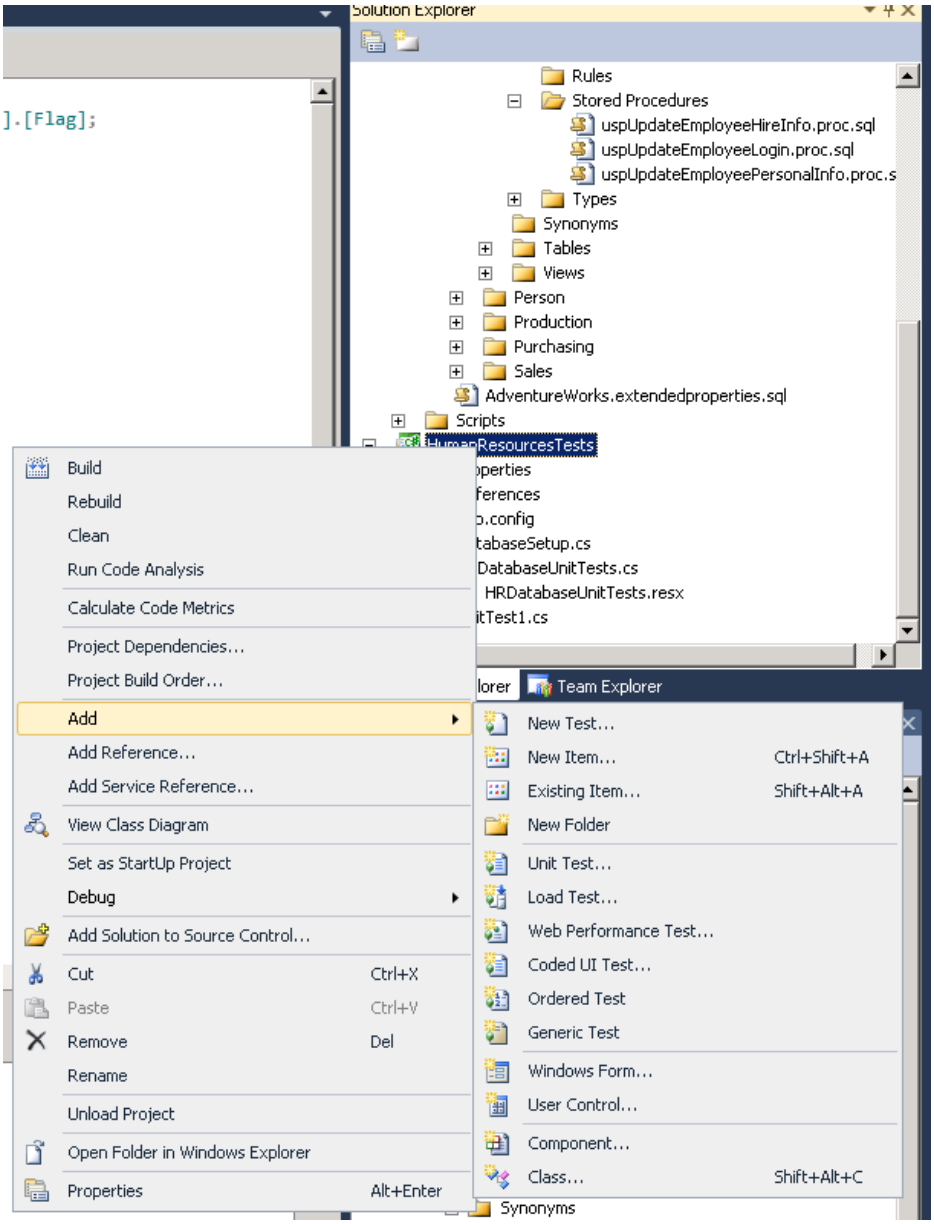


図 41: テスト ソリューションに他の種類のテストを追加する方法

データ生成計画とテスト プロジェクト

データ生成計画は、開発者がデータ ジェネレーターを使用して定義したデータに準拠したテスト データの作成に使用します。データ生成計画は、複数のテスト方法を使用したデータベースのテストをサポートする反復可能な方法でデータが作成されるように策定する必要があります。特定のビジネス機能の領域を対象とする単一のデータ生成計画では、次のテストをサポートしている必要があります。

- データ整合性のテスト
- データベース単体テストを使用したデータ アクセスのテスト
- アプリケーションの統合テスト
- ユーザー インターフェイスのブラック ボックス テスト

複数のテスト方法をサポートするデータ生成計画を作成すると、データを適用できる範囲が広がり、データを再利用できるようになり、1 つのデータ生成計画でサポートできるシナリオが増えます。データで複数のテスト ケースとシナリオをサポートしなければならない場合があるので、この手法を採用すると計画が複雑になることがあります。そのため、計画の対象範囲を 1 つの機能領域に限定することが推奨されています。

テスト プロジェクトに関連付けられている app.config ファイルの構成設定を使用すると、データ生成計画をテスト プロジェクトに組み込むことができます。構成セクションのサンプルを次に示します。

```
<DatabaseUnitTesting>
   <DatabaseDeployment DatabaseProjectFileName="¥<プロジェクト フォルダー>¥<プロジェクト名>.dbproj"
Configuration="Debug" />
   <DataGeneration DataGenerationFileName="¥<プロジェクト フォルダー>¥Data Generation Plans¥<plan
name>.dgen" ClearDatabase="true" />
   <ExecutionContext
         Provider="System.Data.SqlClient"
         ConnectionString="Data Source=.;Initial Catalog=<ターゲット データベース>;Integrated
         Security=True;Pooling=False"
         CommandTimeout="30" />
   <PrivilegedContext
         Provider="System.Data.SqlClient"
         ConnectionString="Data Source=.;Initial Catalog=<ターゲット データベース>;Integrated
Security=True;Pooling=False"
         CommandTimeout="30" />
 </DatabaseUnitTesting>
```

テスト プロジェクトの app.config ファイルで DatabaseUnitTesting セクションを定義すると、各テストの実行前に実行されるデータ生成計画に基づいて、ターゲット データベース (<ターゲット データベース>) にアクセスするテストを繰り返し実行できます。

注

1 つのテスト プロジェクトに関連付けられるデータ生成計画は 1 つだけです。この制限により、テスト プロジェクトとデータ生成計画がアプリケーションの 1 つの機能領域を対象とするようになります。この制限は、データ生成計画とテスト プロジェクトに含まれるテストの両方に大きく影響することがあります。

データとテスト プロジェクトの連携方法を決定するときには、データをどのように構造化する必要があるかということと、ここで紹介したようなテスト プロジェクトを構造化する手法について検討してください。データ生成計画の構造化方法とデータ生成計画がテストの実行に与える影響の詳細なガイダンスについては、このドキュメントの後半のデータ生成計画に関するセクションを参照してください。

- Visual Studio データベース単体テストのメリット
  - 簡単に起動して実行できる
  - データを明確にできる
  - 一般的なテストが既に提供されている
  - 複数のテスト シナリオを 1 回のデータベース呼び出しに簡単にグループ化できる
  - SQL を利用することで、データベース開発者にとってコードが記述しやすくなる
- Visual Studio データベース単体テストのデメリット
  - データ ドリブン テストのサポートが組み込まれていない。同一フレームワーク内の異なるアサート手法を直感的に区別しにくいことがあります。たとえば、結果をテストする SQL を使用してアサートを実行する場合もあれば、テスト フレームワークを使用してアサートを利用する場合もあります。
- 参照データベースとテスト データベースの比較
  - データ ソースとして読み取り専用のデータベースを使用した場合のメリットとデメリット
  - テスト前にデータベースを再作成する手法のメリットとデメリット
  - 開発中のデータベースのテストと変更の管理
  - ラベルを使用した既知のバージョンの作成

データ生成と参照データベース

データ生成計画は、データベースを使用する反復可能なテストをサポートするうえで重要な役割を果たします。ただし、データ生成計画は、すべてのシナリオに適しているわけではないので、データ生成計画を有効に活用するには一定の方法で構造化する必要があります。さまざまな種類のデータ生成オプションに加えてカスタム データ ジェネレーターを作成することもできるので、有益なデータ生成計画を初めて作成するときには、さまざまな手法が考えられます。このセクションでは、ローカルのデータベース テストと自動データベース テストの両方でデータ生成計画を最大限に活用する方法についてのガイダンスを提供します。また、データ生成とテストの実行を構造化して個別の実行計画に分ける方法についても解説します。

テストに適した計画の作成

データ生成計画は、テストと一緒に作成した場合に最も開発しやすくなります。テストの作成前にテストで必要なデータを把握するのは難しいことがあります。また、テストを開発するまで確認できない特殊なデータ シナリオが必要になることもよくあります。このような状況に対処する最も効果的な方法は、ビジネス機能とデータベース スキーマの両方に準拠した論理領域にデータ生成計画を合わせることです。

たとえば、金融アプリケーションを開発している場合、顧客管理専用のデータ生成計画、勘定取引専用のデータ生成計画、アプリケーションのメタデータ管理専用のデータ生成計画などを策定する手法が最も簡単な手法となることがあります。この手法では、データの依存関係を最小限に抑えられるので、データ生成計画が管理しやすくなります。また、アプリケーションとデータベースの特定の機能領域を対象とするほとんどのテストに対応できる幅広いデータが提供されます。依存関係を最小限に抑えることを、計画で生成するデータの境界を定義する際の目標にしてください。1 つの計画で 1 つのデータベースに含まれる全データを生成すると、非常に連携が難しくなることがあります。その一方で、データベースが大規模または複雑な場合は十分なデータが生成されるというメリットもあります。

参照データの使用

参照データは、実質的にすべてのデータベースに存在しています。多くの場合、参照データは、データベースの作成時に、データを静的な参照テーブルに書き込むスクリプトを使用して構成されます。参照データは、国名の一覧やデータの種類 (取引の種類、ユーザーの種類など) で構成されています。

通常、参照データは、データベース プロジェクト内のインストール後スクリプトに含まれる、カスタム SQL スクリプトでキャプチャされます。

参照データは、運用データベースでキャプチャされることもあります。新しいデータベースをゼロから作成するのではなく既存のデータベースを操作する場合は、運用データベースから参照データをキャプチャすることがよくあります。

データベースの作成時に、配置後スクリプトなどによって参照データが作成される場合、データ生成計画の実行がデータに影響する場合もあれば、影響しない場合もあります。データ生成計画の実行を定義するときには、データベースの既存のデータを保持するか、全データを削除するかを必ず指定する必要があります。ただし、現時点ではデータを削除するテーブルとデータを保持するテーブルを個別に選択することはできません。データベースのデータを削除する場合でも、トランザクション テーブルのデータを削除するときには、どこかの時点で参照データが必要になることが往々にしてあります。

推奨事項

このようなシナリオをサポートする最も簡単な方法は、ソース データを含む参照データベースも管理することです。参照データを配置後スクリプトでキャプチャする場合、このデータをテスト データベースで再キャプチャするには、データベースを 2 回配置します。

1. 1 回目の配置が必要なのは、テストに必要な参照データの構造 (データベース スキーマ) または機能 (データ コンテンツ) が変更される場合だけです。1 回目の配置では、ローカルまたはネットワーク サーバー上の既知のデータベースの場所を配置先にし、データベースは読み取り専用ソースとして機能する必要があります。

2. 2 回目の配置は、データベースのデータを削除するデータ生成計画と組み合わせて使用します。データ生成計画の一環として、テーブルの必須列でシーケンシャル データ バインド ジェネレーターを使用して、最初のデータベースから参照データを再設定します。このジェネレーターでは、参照データベースのコンテンツを選択して、テスト データベースに参照データを再設定します。

重要

Visual Studio 2010 より前のバージョンでは、ID 列を使用していると、SQL Server に実装されていた ID 列の再設定方法により、生成計画で 1 つずつずれたキーが作成されていました。Visual Studio 2010 では、この問題は解決されています。

運用データや以前のデータのコピーを参照データベースとして使用している場合、1 回目の配置は必要なく、シーケンシャル データ バインド ジェネレーターのターゲットは運用データになります。

推奨事項

シーケンシャル データ バインド ジェネレーターまたはデータ バインド ジェネレーター[34] を使用すると、特定の列を明示的に対象にする SQL ステートメント (Select City From Addresses など) を作成できます。また、単純に Select * From Addresses ステートメントを使用することもできます。この手法を使用すると、データ生成計画では、生成される列が自動的にフィルター処理され、"ターゲット列の強制変換が可能なデータ型" に限定されます。この手法では、やや簡単かつ迅速に SQL ステートメントを作成できるようになりますが、ソース テーブルに多数の列や行が含まれていると、データ生成計画の実行パフォーマンスが低下することがあります。

意味のあるデータとランダム データの作成

ランダム データを生成するときには、Visual Studio で提供されているデータ ジェネレーターを活用することをお勧めします。生成されたデータは意味を成さないこともありますが、たいては、さまざまなデータ コンテンツが作成されるだけでなく、値の制限 (文字列の長さや数値の処理能力など) を課すことができます。このような値を使用すると、ビジネスで使用する値では検出できない多いバグが検出されることがあります。

多くの場合、データベースのコンテンツは、整合性が保たれていれば問題になりません。たとえば、特定のテストの目的が、データを選択して、既知のデータベース コンテンツに基づいて正しいデータかどうかを確認することである場合、実際の値は問題になりません。このような場合、データ生成計画で設定される既定値で十分に対応できます。しかし、多くの場合は、データが特定の形式になっていることを確認する必要があります。AdventureWorks データベースを例に説明すると、このデータベースでは、多数の列で、その列に含めることができるデータ型についての制約が設定されています (Gender 列の M または F という文字に関する制約などは、その一例です)。このような場合は、2 つの手法を使用できます。1 つは、データ生成計画から制約がある列を除外する手法です。これは簡単ですが、要件に対応できないこともあります。もう 1 つは、既定の文字列ジェネレーターではなく、正規表現ジェネレーターを使用する手法です。正規表現を使用すると、その列に設定できるコンテンツを指定する簡単な手段が提供されます。

他にも、データ バインド ジェネレーターや指定した方法でデータを作成するカスタム ジェネレーターなどを使用する手法があります。カスタム ジェネレーターについては後で説明します。

[34] http://msdn.microsoft.com/ja-jp/library/dd193262.aspx

実際のデータと匿名データの組み合わせ

運用環境で使用されている実際のデータは構造とリレーションシップを含んでおり、シミュレーションが難しいながらも堅牢なテストには欠かせない経験則によるテストをある程度再現できます。ですが、実際のデータには、運用環境の外部からアクセスすべきではない機密情報が含まれていることもあります。

機密情報を削除した運用データのコピーと生成されたデータを組み合わせると、非常に複雑で包括的なテスト データを実現できます。また、データ バインドされた列と生成されたデータの列を組み合わせて使用すると、データのセキュリティを維持しながら、実際のデータの品質と構造に近い状態を再現できます。

パフォーマンス テストとロード テスト

パフォーマンス テストとロード テストは、このガイドの対象範囲外ですが、データ生成計画を利用して大量のデータを非常に効率的に生成することが可能であり、生成したデータを使用すると、データベースを作成して指定した量のデータを設定できることに留意してください。データの品質を確認したり、特定の方法で作成されたデータを使用したりする他のテストと異なり、データ生成計画ではデータの量を重視する必要があります。通常、この種の生成計画は他のテストでは使用されず、他のテストと同じデータ品質を維持する必要はありません。

データ生成計画を使用して大量のデータを生成すると、インデックスの変更やクエリの最適化による影響を特定するテストを実行する場合に特に役に立ちます。この手法により、データベースやクエリの構造が変化してもデータが変更されることはありません。

テストのビルドへの統合

テストをビルド システムに統合する方法の詳細については、このガイドの「Studio データベース プロジェクトでビルドと配置の自動化」で、特に「チーム ビルドを作成する方法」を参照してください。

データ生成計画を使用すべきでない場合

- システムで生成されたデータはわかりにくいことがあるので、データ生成計画は、ユーザーが手動で行うエンド ツー エンドのブラック ボックス テストに適していないことがあります。
- データ生成計画は、循環参照を含むデータベース構造ではうまく機能しません。たとえば、Employee テーブルで Department テーブルを参照している場合に、Department テーブルでマネージャーを特定するために Employee テーブルを参照している場合は循環参照になります。

カスタム データ ジェネレーター

カスタム データ ジェネレーターは、要件を満たすのに必要な任意の種類のデータ コンテンツを生成する機能を備えた拡張メカニズムです。既定の自動テストまたは手動テストで特定の要件を満たす必要があり、この要件を既定のジェネレーターや参照データベースで満たせない場合は、カスタム ジェネレーターを使用します。カスタム ジェネレーターは、柔軟で構成可能になるように開発して、開発者のワークステーションやビルド サーバーなど、さまざまな環境での実行をサポートするようにします。

次のシナリオに当てはまる場合は、カスタム ジェネレーターの使用を検討します。

- サンプル データを提供する、サービス データ ソースやサポートされていないデータ ソースが存在する
- 異なるデータ ソース間、あるいは複数のテーブル、列、またはプロジェクト間でカスタム ジェネレーターを再利用できる

カスタム データ ジェネレーターの開発方法に関する詳細な手順については、MSDN の記事「データ ジェネレーター機能拡張の概要[35]」を参照してください。

配置の検証

ビルド サーバーは、サーバー ベースのビルドとテストの実行に加えて、データベースを既知の場所に配置して配置スクリプトの正確性を検証するように構成されている必要があります。ビルド サーバーでは、VSDBCMD コマンド ライン ツールのスキーマ比較機能を利用するか、データベース プロジェクトのプロパティを使用します。

配置前検証を実行するには、/p:VerifyDeployment スイッチを使用して、アップグレード配置の試行時にターゲット データベースで問題が発生しないことを確認します。

[35] http://msdn.microsoft.com/ja-jp/library/aa833172.aspx

データベースを配置したら、配置したデータベースをデータベース プロジェクト スキーマと比較して、配置したデータベースが想定している結果と一致することを確認できます。このような比較を行う場合は、DropObjectsNotInSource フラグが設定されているかどうかよく確認してください。このフラグが設定されていない場合、データベース プロジェクト モデルで表されなくなったオブジェクトがデータベースから削除されていないと、スキーマ比較で等値比較が失敗することがあります。このフラグは、プロジェクトの配置オプション プロパティ ページを表示するか、/p:DropObjectsNotInSource コマンド ライン スイッチを使用して確認できます。

これらの手法は、自動ビルドやテスト ルーチンで使用したり、手動配置の検証に使用したりすることができます。

データベース スキーマを比較する方法の詳細については、MSDN の記事「方法: データベース スキーマを比較する[36]」を参照してください。

[36] http://msdn.microsoft.com/ja-jp/library/aa833435.aspx

モデルのずれの把握

データベース開発者は、データベース プロジェクトのバージョンを 1 つに限定する必要があることを知っています。とは言え、多くの開発者は、運用環境にデータベースの変更内容を配置する際に想定していたスキーマが見つからなかったときの状況を思い返すことでしょう。この状況が発生する原因はさまざまですが、最もよくある原因は、データベースの管理者やデータベース操作によって独自にデータベース プロジェクトが変更されたことです。データベース管理者は、データベースを制御できるので、更新プログラムの適用、追加のインデックスの適用、データベース スキーマの細かい変更などの作業を行いがちです。

問題は、このような作業と並行して、開発者が、テスト、メンテナンス期間の計画、およびデータベースをアップグレードするための正常性チェックを実行していることです。たとえば、以前に配置したバージョン 1 に基づいた、バージョン 1 + 1 などの計画を立てているとしましょう。データベース管理者がバージョン 1 を変更すると、バージョン 1* が運用環境のマスター データベースになります。開発者が運用データベースにバージョン 1 + 1 のデータベース スキーマを配置しようとすると、テスト時には想定していなかった追加の処理を配置エンジンが実行する場合があります。この場合、アップグレードの配置が失敗するか、インデックスを削除して再作成する必要があるため予想以上に時間がかかります。

メモ

データベース プロジェクトの新しい配置エンジンを使用すると、モデルのずれに配慮しなければならない度合いが少し低くなります。と言うのも、実際に配置する直前に運用データベースと指定のモデルを比較して、差分配置スクリプトが作成されるからです。このため、変更を直接防止し、配置スクリプトを調べてから適用するのが容易になっています。それでもモデルのずれが重大な問題であることに変わりはなく、データベースに実際に配置する前に特定する必要があります。

配置エンジンである VSDBCMD には、ターゲット データベースに作成されていても (DropConstraintsNotInSource オプションと DropIndexesNotInSource オプションの設定が既定で true に設定されているため) ソース データベースには存在しない、既存のインデックスや制約を削除する新しい配置オプションが追加されました。ただし、データベース管理者が自分で調整したインデックスが削除されている状況を目の当たりにして混乱するのを防ぐ必要はあるでしょう。そのため、配置を計画する前にモデルの相違点を把握し、データベースに変更を適用できなくなる停止期間を管理者に通知することをお勧めします。停止期間の開始時に、管理者に VSDBCMD コマンドとインポート機能を使用してモデルを作成して、そのモデルを提供してもらいます。データベース プロジェクトのデータベース スキーマ比較機能を使用して、プロジェクトの変更点をインポートする必要があるかどうかを判断し、マージしたデータベース プロジェクトに対してもテストを実行できることを確認します。

推奨事項

配置作業のマイルストーンについてデータベース管理者と話し合い、テストの作成対象となるモデルが運用環境で使用中のモデルとほぼ同じになるようにします。このようにすると、配置時に問題が発生するリスクを抑えて、明快で正確な配置時の計画を立てることができます。

実際に配置するときには、管理者から提供されたモデルを現在の運用データベースと再度比較します。最初の比較以降に変更が加えられていてモデルにずれがある場合は、次の手順について話し合い、変更点を (.sql 差分スクリプトを生成するなどして) データベースに適用する際のリスクを見積もります。

この作業は、配置戦略とその成功に関して非常に重要で必要不可欠な作業です。

配置のロールバック

配置オプションとしてトランザクション配置を選択すると、既存のデータベースで配置をロールバックできる場合があります。残念ながら、ファイル グループやデータベースのプロパティの変更など、SQL Server のステートメントにはトランザクション ステートメントでないものもあるので、ロールバックできない場合もあります。また、作成前にデータベースを削除すると、トランザクション スコープ内で元のデータベースに戻すことはできません。ロールバックにかかる時間よりも短時間でデータベースのバックアップを復元できる場合は、大規模なデータの移動処理をまったくロールバックしない状況も考えられます。

配置の失敗時に元のデータベースが必要な場合は、実際に配置する前に運用データベースをバックアップしておくことをお勧めします。Visual Studio のツールを使用して、VSDBCMD の配置フラグ (/p:BackupDatabaseBeforeChanges) を true に設定すると、このようなバックアップを実行できます。この設定は、データベース プロジェクトでプロジェクトの配置オプション プロパティ ページを表示するか、VSDBCMD でコマンド ラインのスイッチを使用すると確認できます。

VSDBCMD で使用できるプロパティの詳細については、MSDN の記事「VSDBCMD.EXE コマンド ライン リファレンス (配置およびスキーマのインポート)[37]」を参照してください。

[37] http://msdn.microsoft.com/ja-jp/library/dd193283.aspx

参考資料

- How Do I: Generate Test Data using Visual Studio Team System Database Edition?
  (http://msdn.microsoft.com/en-us/teamsystem/cc501309.aspx、英語)
- テスト駆動開発をデータベース プロジェクトに適用する
  (http://msdn.microsoft.com/ja-jp/magazine/cc164243.aspx)
- Database Unit Testing with Team Edition for Database Professionals
  (http://msdn.microsoft.com/en-au/bb381703(VS.80).aspx、英語)
- Scott Ambler 著『Agile Database Development』
  (http://www.ambysoft.com/books/agileDatabaseTechniques.html、英語)

付録

データベース プロジェクトのトレースの有効化

Visual Studio の製品ディレクトリにある reg ファイルを実行すると、データベース プロジェクトのトレースが有効になります。トレースを有効にするには、次の手順を実行します。

1. 製品ディレクトリにあるレジストリ ファイル (%Program Files%¥Microsoft Visual Studio 10.0¥VSTSDB¥Tracing.reg) を開きます。
2. ファイルを編集し、編集したファイルのコピーを一時ディレクトリに保存します。TraceSwitch を 00000001、TraceToLogFile を 00000001、DisplayCallStack を 00000001、LogDir を "c:¥¥Temp¥¥VSTSDB" に設定します。
3. 一時ディレクトリにある新しいファイルをダブルクリックして、エントリをレジストリに追加します。
4. Visual Studio を起動している場合は再起動します。Visual Studio の次のインスタンスで一連のログが有効になるには、Visual Studio のインスタンスをすべて終了する必要があります。

データベース プロジェクトを使用したセッションごとに、ログ ディレクトリとログが追加されます。エラーが発生すると、スタックを含め、そのエラーの詳細なログ エントリが記録されます。

レジストリに追加する前の Tracing.reg ファイルの内容は、次のとおりです。

Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER¥Software¥Microsoft¥VisualStudio¥10.0¥VSTSDB¥Tracing]

"LogDir"="c:¥¥Temp¥¥VSTSDBTRACE"

"PrefixTime"=dword:00000001

"PrefixPid"=dword:00000001

"PrefixThreadId"=dword:00000001

"TraceSwitch"=dword:00000001

"TraceToLogFile"=dword:00000001

"TraceToDebugOutput"=dword:00000000

"DisplayCallStack"=dword:00000001

"UniqueLogFile"=dword:00000001

"EventIdsToTrace"=dword:0001ffff

参考資料

全般

Visual Studio ALM Rangers のサイト

http://msdn.microsoft.com/en-us/teamsystem/ee358786.aspx (英語)
http://www.tinyurl.com/almrangers (英語)

Visual Studio Widgets

http://www.teamsystemwidgets.com (英語)

Team System のビデオ

http://msdn.microsoft.com/en-ca/vsts2008/bb507749.aspx?wt.slv=topsectionsee (英語)

MSDN Web サイト

http://msdn.microsoft.com/ja-jp/default.aspx